НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Методи навчання з підкріпленням в комбінаторних іграх двох учасників»

Виконав:
студент IV курсу, групи КА-61
Єлісєєв Владислав Вікторович _____

Керівник:
доцент, д.ф.-м.н, доцент кафедри ММСА
Ігнатенко Олексій Петрович _____

Консультант з економічного розділу:
доцент, к.е.н., доцент кафедри ТТПЕ
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
доцент, к.т.н., доцент кафедри ММСА
Коваленко Анатолій Єпіфанович _____

Рецензент:
професор, д. ф.-м.н., зав. відділу ІПС НАНУ
Дорошенко Анатолій Юхимович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 "Системний аналіз"

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Єлісєєву Владиславу Вікторовичу

1. Тема роботи «Методи навчання з підкріпленням в комбінаторних іграх двох учасників», керівник роботи Ігнатенко Олексій Петрович, д.ф.-м.н., доцент, затверджені наказом по університету від «25» травня 2020 р. №1143-с

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи _____

_____

4. Зміст роботи _____

_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

_____

_____

6. Дата видачі завдання «03» __04___ 2020 р.

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О.А. доцент		

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.			
2.			
3.			
4.			
5.			

Студент Владислав ЄЛІСЄЄВ

Керівник Олексій ІГНАТЕНКО

РЕФЕРАТ

Дипломна робота: 101 с., 43 рис., 6 табл., 2 додатки, 32 джерел.

АГЕНТ, ВИГРАШНА СТРАТЕГІЯ, ГЛУБОКЕ НАВЧАННЯ, КОМБІНАТОРНА ГРА, МАРКОВСЬКИЙ ПРОЦЕС ПРИЙНЯТТЯ РІШЕНЬ, НАБЛИЖЕНА ОПТИМІЗАЦІЯ СТРАТЕГІЇ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Об’єктом дослідження є комбінаторні ігри двох осіб з повною інформацією, де складність пошуку стратегії полягає у великий кількості стратегій та траєкторій гри, що ускладнює пошук оптимальної стратегії.

Метою дослідження є огляд останніх досягнень науки у сфері навчання з підкріпленням, розгляд можливості застосування методів навчання з підкріпленням до задач пошуку стратегій гри у комбінаторних іграх.

Предмет дослідження: методи глибокого навчання з підкріпленням для вирішення задачі пошуку виграшної стратегії у комбінаторних іграх та їх застосування на конкретних прикладах.

Дослідження ґрунтується на наукових публікаціях та інших матеріалах закордонних конференцій та архівів в галузі машинного навчання, глибокого навчання та глибокого навчання з підкріпленням і пошуку виграшних стратегій в іграх.

Програмна реалізація моделі та інфраструктури навчання написана мовою Python, з використанням фреймворку TensorFlow та OpenSpiel.

ABSTRACT

Thesis work: 101 pp., 43 fig., 6 tabl., 2 app., 32 sources.

AGENT, WINNING STRATEGY, DEEP LEARNING, COMBINATORIAL GAME, MARKOV DECISION-MAKING PROCESS, APPROXIMAL STRATEGY OPTIMIZATION, REINFORCEMENT LEARNING

The object of the study is combinatorial games of two people with complete information, where the difficulty of finding a strategy lies in the large number of strategies and game trajectories, which complicates the search for the optimal strategy.

The purpose of the study is to review the latest advances in science in the field of reinforcement learning, to consider the possibility of applying methods of reinforcement learning to the tasks of finding game strategies in combinatorial games.

Subject of research: methods of deep learning with reinforcement to solve the problem of finding a winning strategy in combinatorial games and their application on specific examples.

The research is based on scientific publications and other materials of foreign conferences and archives in the field of machine learning, deep learning and deep reinforcement learning and winning strategy optimization in games.

The software implementation of the model and learning infrastructure is written in Python, using the TensorFlow and OpenSpiel frameworks.

ЗМІСТ

с.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

RL	– навчання з підкріпленням
MDP	– марковський процес прийняття рішень
ANN	– штучна нейронна мережа
ДП	–динамічне програмування
DQN	–deep Q-network
$U(\cdot)$	–рівномірний розподіл
ММП	–метод максимальної правдоподібності

ВСТУП

Люди здавна грають у різноманітні ігри. Для багатьох розвинених живих організмів гра стала способом навчання, покращення власних можливостей та саморозвитку. Так хижаки через гру навчаються полюванню ще в дитинстві, сучасні педагогічні методики рекомендують змішувати навчальний процес з різними іграми для кращого засвоєння навчального матеріалу. Також ігри стали ознакою високої інтелектуальної розвиненості. Найкращі гравці в шахи або ж у гру ґо, що родом з Китаю, вважаються дуже інтелектуально розвиненими людьми і демонструють можливості людського мозку. Такі ігри стали розглядатися як предмет вивчення і стали викликати увагу математиків лише з початку XX сторіччя.

Початок вивчення було покладено статтею Чарльза Баутона “Нім, гра з повною математичною теорією” від 1902 року. З розвитком математики була розроблена математична теорія комбінаторних ігор, до яких належать ті ж самі шахи та ґо. Впродовж довгого часу саме ці ігри були індикатором розвитку штучного інтелекту і вважалося, що штучний інтелект не може перевершити людський розум, через високу складність цих ігор. Проте в останні 25 років багато що змінилось. Поступово штучний інтелект розвивався і кожна його перемога над людиною стає величезним досягненням для науки в цілому. Особливо вплинув на розвиток штучного інтелекту в напрямку комбінаторних ігор технологічний розвиток комп’ютерних систем, яка в свою чергу надала можливість використання нейронних мереж та підштовхнуло до розвитку глибокого навчання та навчання з підкріпленням. Саме завдяки розвитку цих технологій, зараз ми маємо системи прийняття рішень, які здатні перемагати найкращих гравців.

Метою даної роботи є дослідження останніх досягнень науки у сфері навчання з підкріпленням, застосування підходів навчання з підкріпленням у комбінаторних іграх двох гравців. Результати цієї роботи можуть бути застосовані для практичної розробки і побудови нових комбінаторних ігор.

У першому розділі розглянуто особливості предметної області – комбінаторних ігор. Наведено приклади таких ігор, розглянуто їх класифікацію, висвітлена актуальність дослідження цієї області та розглянуто класичні підходи теорії до задачі.

У другому розділі детально розглянуто теорію навчання з підкріпленням, марковських процесів прийняття рішень, динамічного програмування. Також була висвітлена проблема дослідження в навчанні з підкріпленням. Наведено класифікацію алгоритмів навчання з підкріпленням та ретельно розібрано деякі із них.

У третьому розділі розглянуто особливості програмної реалізації для дослідження задачі пошуку стратегії у комбінаторних іграх методами навчання з підкріпленням. Надано результати дослідження на прикладі середовища гри Прорив.

У четвертому розділі наведене економічне підґрунтя щодо методів розробки програми для навчання агентів-гравців в комбінаторні ігри. Проведена кількісна і якісна оцінка параметрів, економічний аналіз варіантів розробки програмного продукту.

ПОСТАНОВКА ЗАДАЧІ

В роботі поставлено наступні задачі:

дослідити наявні методи пошуку виграшних стратегій у комбінаторних іграх;

розглянути можливість застосування методів глибокого навчання з підкріпленням до задачі пошуку виграшної стратегії у комбінаторних іграх;

визначити методи, придатні для застосування у даній задачі, і застосувати їх на прикладі конкретної комбінаторної гри.

Розділ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Комбінаторні ігри.

Теорія комбінаторних ігор, це математична теорія, що вивчає ігри двох осіб, в яких в кожен момент гри є стан, яку гравці по черзі замінюють певним ходом, аби досягти умови перемоги. В цій теорії не вивчаються ігри, пов’язані із випадковостями (наприклад гра “Покер”), а лише ті ігри, в яких позиція та всі можливі ходи відомі обом гравцям.

Розглянемо деякі визначення та концепти, які використовуються у теорії комбінаторних ігор [1][2]. У комбінаторних іграх приймають участь двоє осіб, що приймають рішення, яких будемо називати гравцями (або ж агентами).

У іграх є позиції (або стани), в яких гравці можуть виконувати дії, що призводять до нових станів. Набір усіх можливих станів у грі позначаємо S. Гравець впродовж гри може виконати дію, яка може змінити поточний стан на новий стан відповідно до деякої моделі переходу. Послідовна гра - це гра, в якій гравець обирає дію, перш ніж інший гравець (гравці) обирає свою дію. Важливо, щоб гравці мали певну інформацію щодо ходів, які виконували попередні гравці, інакше різниця у часі не впливає на стратегію (такі ігри називаються одночасними). Кроком позначимо акт виконання дії (в деякому стані). Комбінаторні ігри - це підмножина послідовних ігор для двох гравців, в яких немає елемента випадковості, і кожен гравець має досконалу інформацію. Досконала інформація означає, що обидва гравці мають повну інформацію щодо дій, які виконує інший гравець. Ігри, в яких немає елемента випадковості, називаються детермінованими.

1.2 Гра “Нім”

Гра, яка поклала початок розвитку теорії комбінаторних ігор стала гра “Нім”. Нім - одна із найстаріших відомих комбінаторних ігор. Гра, схожа на сучасний Нім, була відома в Китаї ще тисячоліття тому. Перші згадки про дану

гру в Європі з’явилися в XVI ст. Сучасну назву грі дав математик Чарльз Л. Баутон - професор Гарвардського університету, який також розробив суцільну теорію навколо цієї гри [3]. Теорія гри Нім стала основою теорії комбінаторних ігор в цілому. На рисунку 1.1 зобразимо приклад гри Нім.

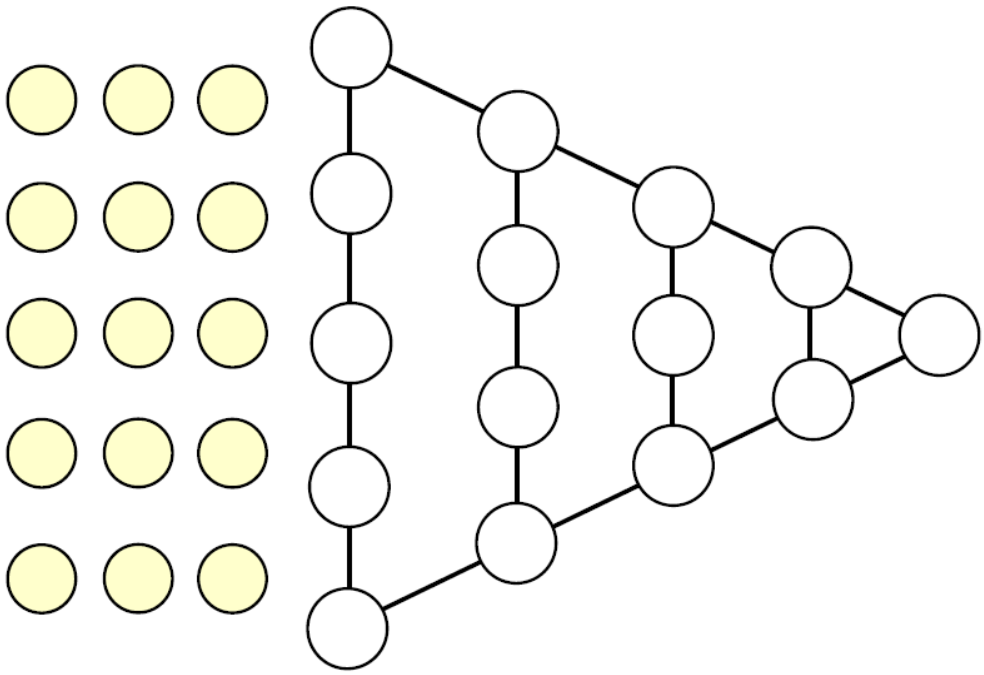


Рисунок 1.1 – Класичний приклад комбінаторної гри – гра “Нім”

Правила гри Нім дуже прості, два гравці по черзі видаляють ("nimming") об’єкти з різних окремих кучок. Кожного кроку гравець має видалити хоча б один об’єкт, проте він може видалити будь-яку кількість об’єктів, якщо вони всі належать одній і тій самій купці. В залежності від виду гри, ціллю може бути або взяти останній об’єкт з усіх куч, або ж аби протилежній гравець забрав останній предмет.

У випадку на рисунку 1.1 ми маємо 5 кучок, в яких знаходиться 15 предметів. Зазвичай позиції в грі Нім записують у вигляді розмірів наявних у грі кучок, так, наприклад, початкова позиція у грі вище записується як (1,2,3,4,5).

1.3 Класифікація комбінаторних ігор

Розглянемо, які є види комбінаторних ігор. В комбінаторній теорії ігор вводяться такі важливі види ігор як, упереджена та неупереджена гра.

Неупереджена гра - це комбінаторна гра, яка має такі властивості:

- дія, яка доступна певному гравцю залежить лише від стану гри, але не залежить від того, крок якого гравця зараз;

- гравець який не має можливості здійснити хід програє;

- гра завжди має мати кінець.

Умову, що стверджує як закінчується гра можна замінити, аби отримати так звану гру у мізер. Властивості гри в мізер:

- дія, яка доступна певному гравцю залежить лише від стану гри, але не залежить від того, крок якого гравця зараз;

- гравець який зробив останній хід програє;

- гра завжди має мати кінець.

Упередженою або ж партизанською грою називається така комбінаторна гра, в яка не є неупередженою чи грою у мізер. Тобто якщо існує стратегія, яка призводить до того, що гра не завершується, чи якщо дії гравця залежать від того який зараз крок, то така гра є партизанською.

Так, наприклад, ігри з гральним кубиком не є комбінаторними іграми, оскільки в них присутня стохастична складова. Також багато картярських ігор не є комбінаторними, оскільки в них треба перемішувати колоду карт. Прикладом неупередженої гри є математична гра Нім, в якій два гравці беруть спільні предмети по черзі, які розкладені на декілька груп. Оскільки предмети спільні для обох гравців - то стан гри не залежить від того, крок якого гравця зараз, а лише від кількості предметів в купках. На відміну від гри Нім, такі відомі ігри як шахи, шашки або ж го є упередженими, оскільки гравці ходять власними фігурами, проте все ще є комбінаторними.

Також комбінаторні ігри можна класифікувати ігри за їх формою, тобто по їх матеріальним ознакам:

1. ігри на дошці: ігри такого виду використовують дошку, на якій зазвичай відбувається вся гра. Зазвичай дошка ділитися на певні позиції, які і становлять основу різних станів. Прикладом такої гри є гра Шахи та гра Ґо.
2. ігри з фішками: гравці виконують свої дії над фішками, які можуть бути як спільними для обох гравців, так і різного виду для обох гравців. Приклад - гра 4 в ряд.
3. ігри з картами: в цьому виді ігор гравці виконують дії за допомогою пластикових, дерев'яних, паперових або карт, або ж власне самі карти і є основою для гри. Карти не обов'язково мають бути гральними.
4. ігри з монетами: в цьому випадку монети часто відіграють роль фішок, гравці виконують дії над монетами, аби досягти перемоги. Дуже поширеним підвидом ігор з монетами є ігри з перевертанням монет.
5. ігри з папером та ножицями та\або з олівцями: можна придумати безліч власних варіантів ігор використовуючи папір та власну уяву.

1.4 Актуальність дослідження комбінаторних ігор

Комбінаторні ігри відомі людству з давніх давен. Комбінаторними іграми є такі всесвітньо відомі ігри як Ґо (~4000 років тому, Китай), шахи (VI ст. н.е., Індія), шашки, японські шахи сьогі тощо. Ці ігри залишаються популярними і на сьогодні для гравців по всьому світу.

Комбінаторні ігри часто мають відносно прості правила, завдяки чому вони часто набувають популярності.

Також комбінаторні ігри є дуже привабливими з точки зору розвитку штучного інтелекту. Через їх складність, величезну кількість можливих станів та можливих дій, на сучасному етапі розвитку інформаційних технологій неможливо побудувати напряму систему прийняття рішень, яка б брала до уваги всі комбінації та можливі дії гравців. Ще донедавна вважалося, що

штучний інтелект не може перемогти людину в силу цих факторів. В 1997 році програма Deep Blue [4], розроблена IBM, перемогла чемпіона світу з шахів Гарі Каспарова. Ця подія стала значним поштовхом для розробки штучного інтелекту. Наступним етапом у розвитку штучного інтелекту стали спроби перемогти людину у грі го, оскільки вона має набагато більше комбінацій.

Британська компанія DeepMind представила у 2015 році алгоритм AlphaGo [5], AlphaGo Zero [6], яка змогла перемогти на той час чемпіона у го – Лі Седоля. Всі ці алгоритми засновані на методі під назвою алгоритм пошуку Монте-Карло на деревах (Monte Carlo Tree Search) [8].

Із стану алгоритм іде по певній із гілок дерева гри донизу, і в кінці оцінює цю гілку. Так дерево гри інкрементально добудовується і цей алгоритм не потребує повного розкриття дерева гри.

Крок перший, Selection: у нас є дерево позиція, і алгоритм кожен раз робить крок, обираючи найкращий дочірній вузол для поточної позиції. Крок другий, Expansion: припускається, що алгоритм досяг кінця дерева, але це ще не кінець гри. На кінці цієї гілки створюється стан, до якого алгоритм спускається по дереву. Крок третій, Simulation: новий вузол у дереві відповідає, фактично, ігровій ситуації, в якій ми опинилися вперше. Ця ситуація оцінюється методом rollout: грається партії з поточної позиції і перевіряється, скільки було виграних партій. Одержаний результат і вважається оцінкою вузла. Крок четвертий, Backpropagation: йдемо вгору по дереву і змінюємо ваги всіх батьківських вузлів в залежності від того, наскільки нова вершина гарна. Фактично, кожен вузол на дереві зберігає два значення: оцінку (value) поточної вершини і кількість разів, скільки ми її відвідали. Це і є один крок алгоритму пошуку Монте-Карло на деревах.

Алгоритми DeepMind користуються схожим методом, проте мають багато ускладнень, як наприклад оцінювання вершин нейронними мережами, пошук Монте-Карло на деревах в інформаційних просторах (ISMCTS) [9] тощо. В подальшому була розроблена ще більш загальна версія їх моделі [7], яка здатна грати також у ігри типу шахи та сьогі.

Зображення матчу проти чемпіона світу з ґо (рисунок 1.2):

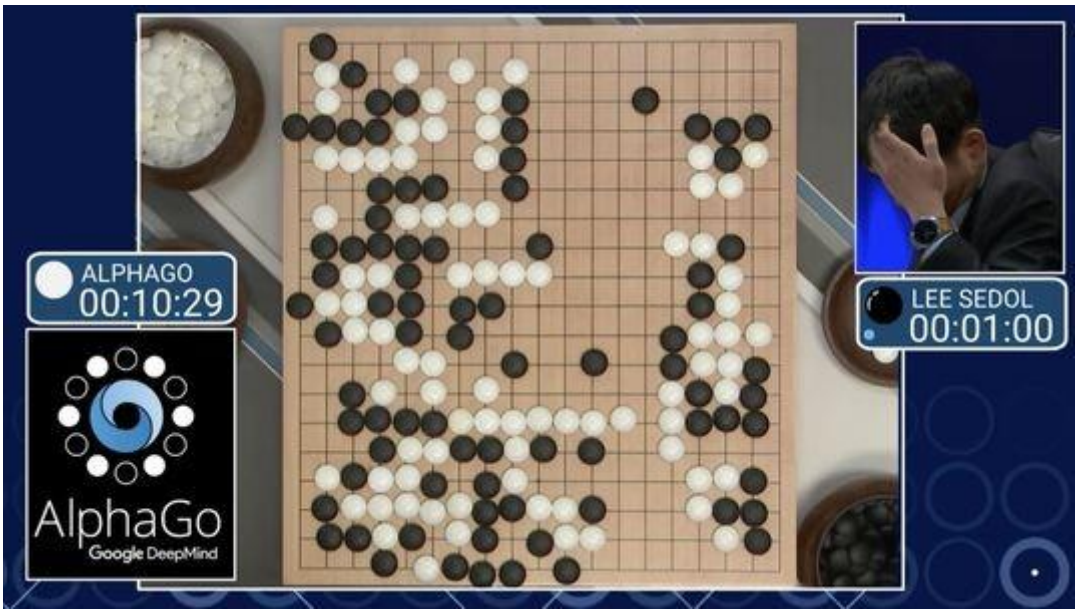


Рисунок 1.2 – Матч програми AlphaGo проти людини - професійного гравця Лі Седоля

Таким чином, задача пошуку стратегій в комбінаторних іграх є актуальною на сьогодні, як дослідницька задача, для розвитку технологій штучного інтелекту а також має прикладне значення для розробників таких ігор.

1.5 Класичні методи вирішення

Застосування теорії комбінаторних ігор полягає в тому, що треба визначити оптимальну послідовність дій для обох гравців аж до самого кінця гри, і таким чином визначити оптимальний хід у кожному стані. Однак, на практиці майже не можливо реалізувати такі програми, через надмірну складність та велику кількість комбінацій, або ж це тільки можливо для дуже простих ігор.

Розглянемо комбінаторних гру наступного виду [10]. Є кінцевий безконтурний орієнтований граф G і вершинах в ньому; у зазначеній вершині

знаходиться фішка; двоє гравців по черзі рухають її по дугам графа; за один хід можна пересунути фішку з однієї вершини в суміжну з нею вершину уздовж дуги, яка їх з'єднує; хто не може зробити хід, програє. Вершину графа, в якій знаходиться фішка, будемо називати позицією ігри. Таким чином, в графі $G = (X, F)$ множина X містить всі можливі позиції, а $F(x)$ являє собою ті позиції, в які гравець може перейти з $x \in X$. Будемо називати елементи $F(x)$ наслідниками, або, краще, опціями позицій. Якщо $F(x)$ порожньо, тобто позиція немає опцій, x називається термінальної позицією. Таким чином, гра на графі $G = (X, F)$ розігрується, як тільки зафіксована початкова позиція $x_0 \in X$, відповідно до правил:

1. грають 2 гравця, 1-й гравець ходить першим, починаючи в позицій;
2. ходи гравців чергуються;
3. в позиції x гравець, чия черга ходити, повинен вибрати позицію $y \in F(x)$;
4. гравець, чия черга ходу припала на термінальну позицію, програє.

Граф G може бути і нескінченним. У цьому випадку, щоб гра мала властивість закінчення (тобто щоб уникнути нескінченного числа ходів), вимагають виконання більш сильного умови, ніж відсутність контурів прогресивної обмеженості: для будь-якої позиції $x \in X$ знайдеться натуральне n таке, що будь-який шлях, що починається в x, має довжину n.

Відповідно до описаних вище правил, гра на графі це нормальна неупереджена комбінаторна гра (без нічиїх). Класичним підходом до розв'язку комбінаторної гри є застосування дерева цієї гри для пошуку у ньому найкращої стратегії для обох гравців.

1.6 Висновки до розділу 1

В цьому розділі були розглянуті комбінаторні ігри, їх класифікація, актуальність та проблематика в даній теорії. Також було розглянуто класичний теоретичний підхід. Було дане визначення комбінаторних ігор,

повної інформації, визначення неупередженої та партизанської гри. Проведений огляд історії комбінаторних ігор, показана їх актуальність та важливість їх дослідження для методів штучного інтелекту. Також було розглянуто останні досягнення науки щодо побудови систем прийняття рішень в комбінаторних іграх.

Розділ 2 МЕТОДИ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

2.1 Задача навчання з підкріпленням

2.1.1 Вихідні поняття

Машинне навчання – це інженерна та наукова дисципліна, яка є підрозділом штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися. Машинне навчання складається з трьох основних напрямків, навчання з учителем, навчання без учителя, та предмет дослідження цієї роботи – навчання з підкріпленням. Навчання з підкріпленням уособлює собою метод навчання, коли агент нагороджується або карається за певні дії, які він виконує. Цей метод покладається на евристику, що за позитивних нагород агент зможе розпізнати оптимальну стратегію і більш імовірно почне її повторювати у середовищі, а при негативних нагородах, або покараннях, він зможе уникати дану стратегію в майбутньому. RL не можна віднести до двох інших парадигм машинного навчання через низку суттєвих відмінностей [11].

Особливості постановки задачі навчання з підкріпленням:

На відміну від навчання з учителем, в навчанні з підкріпленням у постановці задачі немає правильних відповідей. Це означає, що немає вихідних даних, за якими будують модель, аби вона відповідала цим даним. У випадку навчання агенту на винагороду можуть впливати стан середовища, його теперішні та минулі дії тощо. Також функція винагороди при навчанні з підкріпленням може бути неоднорідною. Так в багатьох іграх (у тому числі комбінаторних), як наприклад Шахи, логічною винагородою для агенту є 1, якщо він виграв у грі та -1, якщо програв. Проте агент може отримати дану нагороду від середовища лише тоді, коли він завершить гру, тобто він не має інформації про ефективність своїх дій аж до самого завершення гри. Іноді можна вводити певні евристики, за допомогою яких можна надавати більш однорідні нагороди агенту. У RL під час навчання агенту, алгоритм не має інформації щодо правильності виконання самих дій, а лише про оптимальність

власне траєкторії агенту у середовищі. Тобто алгоритм не дає напряму відповіді на питання які дії треба обрати.

2. Навчання без учителя відрізняється тим, що у ньому алгоритми не надають взагалі ніяких відповідей, або майже зовсім. Задачі, які розв'язуються методами навчання без учителя часто є генеративними або задачами кластеризації чи зменшення розмірності. У навчанні з підкріпленням відбувається пошук певної стратегії для певного середовища, а в навчанні без учителя алгоритми відшукують певну структуру даних або залежність, яка не є заданою або припущеною, як в навчанні з учителем. Результати навчання без учителя часто використовують для певного подальшого дослідження.

Особливістю навчання з підкріпленням є те, що безпосередньо у алгоритмі навчання приймають участь агент і середовище. Схему їх взаємодії можна описати наступним чином [11]:

На кожній ітерації алгоритму середовище генерує стан та надає його агенту спостереження.

Агент, приймає рішення щодо виконання однієї з можливих дій та виконує її безпосередньо у середовищі, впливаючи на нього.

Агент отримує реакцію середовища на його дію у вигляді винагороди та наступне спостереження стану середовища.

Проілюструємо цю схему (рисунок 2.1).

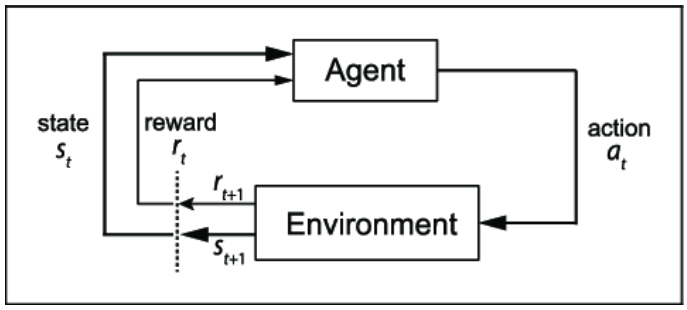


Рисунок 2.1 – Схема взаємодії агенту із середовищем

Стан, який генерується середовищем містить повну інформацію про середовище. Проте спостереження, які отримує агент, дуже часто мають лише

часткову інформацію про середовище. Так, наприклад, комбінаторні ігри є прикладом таких середовищ, де агент отримує повну інформацію про стан гри. Наприклад гравець у Нім повністю спостерігає кількість предметів у купках і має інформацію про те, чий зараз хід. Прикладом середовища, де агент отримує спостереження, що не описують повного стану гри є карткова гра “Покер”. Гравець у Покер, який є агентом, отримує лише інформацію про карти, які він має у себе а також які лежать на столі. Проте повний стан гри “Покер” є також карти, які мають інші гравці.

2.1.2 Формалізація схеми взаємодії агенту та середовища марковським процесом прийняття рішень

Для формалізації взаємодії агенту та середовища часто припускається виконання марковьскої властивості:

$$p(s_{t+1}|s_0, s_1, \dots s_t) = p(s_{t+1}|s_t)$$

Марковська властивість означає те, що майбутні події залежать лише від поточного стану системи чи процесу, і не залежить від минулих станів системи

Дійсно, в багатьох реальних випадках дане припущення є правильним, оскільки стан системи не завжди залежить від усієї історії її існування, а часто лише від поточного стану. Проте у деяких випадках все ж таки ця властивість не є правдивою. Тоді, розглянемо наступне перевизначення стани системи s_t для того, щоб отримати виконанння марковької властивості. Позначимо:

$$s_t = (s_t, s_{t-1}, \dots, s_{t-k})$$

Зазаначимо, що така заміна станів можлива завжди, покладемо $k = t$, таким чином новим станом системи стане вся передісторія станів вихідного процесу.

З практичної точки зору таке переозначення не є доцільним, оскільки воно має очевидний недолік – накопичення історії станів призводить до значних витрат пам’яті та збільшення кількості обчислень всіх нових станів. Для збереження інформації про стан треба використовувати набагато більше ресурсів, а також загалом кількість станів процесу значно зростає.

Більш формально, описаний вище процес можна назвати марковським процесом прийняття рішень [11].

Марковським процесом прийняття рішень називають п’ятірку, що складається з 4 множин та одного числа $\mathrm{M} = (S, A, T, R, \gamma)$ де:

S – множина станів, в яких може перебувати середовище;

A – множина дій, які агент може виконати, може бути як дискретною, так і неперервною;

$P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ –ймовірності переходів зі стану в стан, в залежності від дії агенту. Такі $P_a(s, s')$ формують оператор переходів множини станів T.

Даний оператор можна описати так:

$$\mu_{t,j} = p(s_t = j)$$

$$\xi_{t,k} = p(a_t = k)$$

$$T_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$$

Тоді за формулою повної ймовірності:

$$\mu_{t,j} = \sum_{j,k} T_{i,j,k}\, \mu_{t,j} \xi_{t,k}$$

$R(s, a)$ – середня винагорода, яку отримує агент за виконання дії $a \in A$ в стані $s \in S$. Функція $R(s, a)$ також називається функцією підкріплення.

γ – коефіцієнт дисконтування.

Марковські процеси прийняття рішень можна вважати у деякому сенсі марковськими ланцюгами, а одже вони можуть мати графічне представлення.

Зобразимо граф MDP на рисунку 2.2.

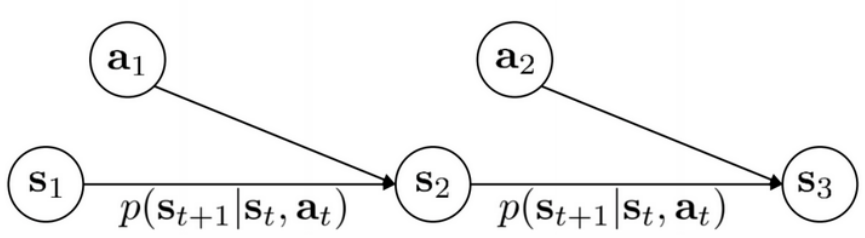


Рисунок 2.2 – Графічне представлення MDP

2.1.3 Узагальнення MDP

Описаний вище марковський процес прийняття рішень був введений для взаємодії агенту і середовища з повною інформацією. Це означає, що агент, що приймає рішення отримує повну інформацію щодо стану середовища. MDP можна узагальнити на випадок, коли агент спостерігає лише частково стан середовища, тобто він не знає всієї інформації про стан s_t. Такі середовища також називаються не повністю спостережними середовищами. Спостереження, які має агент щодо стану s_t позначаються o_t. Прикладом такого середовища можна навести середовище гри “Джин Руммі”. Гравці, які у даному прикладі виступають агентами спостерігають лише карти у своїй власній руці, а також вже зіграні карти та колоду відкинутих карт. У той же самий час вони не спостерігають карти у основній колоді карт а також карти свого суперника (рисунок 2.3).

Рисунок 2.3 Приклад середовища з неповною інформацією

Головною проблемою і особливістю у таких середовищах є те, що можлива ситуація, коли агент спостерігає однакові спостереження середовища, коли насправді середовище знаходиться у різних станах. Це додає особливої складності та унеможливлює використання деяких алгоритмів.

Формально, узагальненим MDP на випадок середовища з неповною інформацією називається $M = (S, A, O, T, E, R, \gamma)$ де:

1. S – множина станів.
2. A – множина дій.
3. O – множина спостережень, які агент отримує від середовища.
4. T – оператор переходів множини станів.
5. E – ймовірність емісії спостережень $p(o_t|s_t)$, де $o_t \in O$.
6. R – функція підкріплення, $R: S \times A \to \mathbb{R}$.

Наведемо графічне представлення MDP (рисунок 2.4).

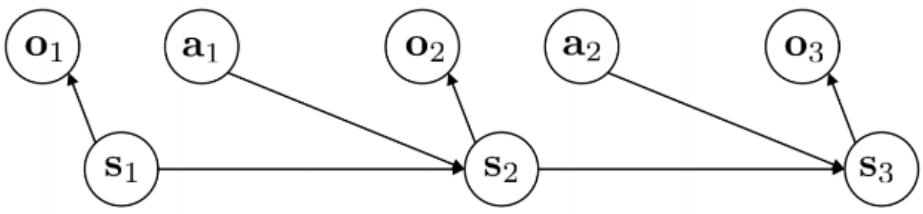


Рисунок 2.4 – Графічне представлення MDP для середовищ з неповною інформацією

2.1.4 Поняття стратегії в термінах MDP

Так як задачею навчання з підкріпленням є пошук оптимальної стратегії агенту в середовищі, то важливо зрозуміти, що саме означає стратегія в термінах марковського процесу прийняття рішень, який є основою для навчання.

Стратегією (policy) називають функцією π, що за станом s визначає, які дії приймає агент, тобто визначає розподіл $p(a|s)$ дій агенту.

Стратегії можуть бути детермінованими та стохастичними. Детермінованими стратегіями називаються такі стратегії π, розподіл яких $p(a|s)$ включає ймовірності 1 та 0.

2.1.5 MDP в задачі навчання з підкріпленням

Марковські процеси прийняття рішень це основний спосіб моделювання прийняття рішень агентів у середовищі, а отже вони є невід'ємною частиною теорії навчання з підкріпленням.

Таким чином, ввівши поняття основні поняття агента, середовища, стратегії, станів, дій та оператора переходу можна ввести поняття траєкторії.

Траєкторія – це послідовність станів та дій $\tau = (s_1, a_1, s_2, a_2, \dots s_T, a_T)$ – скінчена траєкторія, чи $\tau = (s_1, a_1, s_2, a_2, \dots)$ – нескінченна траєкторія.

Стратегія та оператор переходів разом визначають розподіл траєкторій, записаний формулою (2.1):

$$p_\pi(\tau) = p(s_1) \prod_{t=1}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{2.1}$$

В рамках розв'язання задачі навчання з підкріпленням, агенту необхідно знайти стратегію $\pi^*(s)$ (або ж $\pi^*(o)$ у випадку середовища з неповною

інформацією). Така стратегія оптимальна у сенсі максимізації кумулятивної очікуваної винагороди:

$$\pi^* = argmax_\pi \mathbb{E}_{\tau \sim p_\theta(\tau)} \sum_t R(s_t, a_t)$$

Позначимо формулою (2.2) цільовий функціонал цієї задачі, який є очікуваною кумулятивною винагородою:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} R(\tau) \tag{2.2}$$

У випадку скінчених просторів дій була доведена теорема [12] про існування оптимальної стратегії, яка максимізує цільовий функціонал для марковського процесу прийняття рішень, це свідчить про адекватність даного підходу до пошуку стратегій.

Остаточно, MDP у задачі навчання з підкріпленням має такий вигляд (рисунок 2.5):

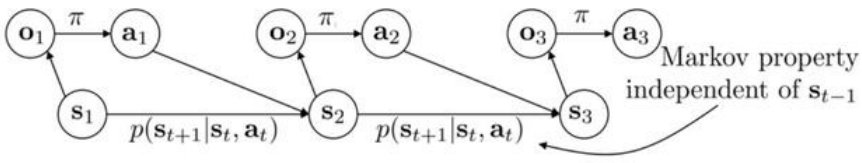


Рисунок 2.5 – Графічне зображення MDP у навчанні з підкріпленням.

2.2 Стратегії та їх апроксимація

У минулому підрозділі було введено поняття стратегії у навчанні з підкріпленням. Стратегії поділяються на детерміновані та стохастичні. Наведемо більш формалізовані визначення цих видів стратегій.

Розглядатимемо більш загальний випадок марковського процесу прийняття рішень з неповною інформацією, оскільки у випадку з повною

інформацією можна вважати, що простори станів S та простір спостережень A співпадають ($s_t = o_t$).

Так, детермінованою стратегією називають невипадкову функцію:

$$\mu: o_t \to a_t,$$

що ставить у відповідність спостереженню $o_t \in O$ дію $a_t \in A$.

Стохастичною стратегією називають випадкову функцію:

$$a_t \sim \pi(\cdot, s_t),$$

тобто дії розподілені за певним випадковим розподілом, який залежить від стану середовища.

Фактично, на практиці для таких стратегій застосовується означення, що було надано раніше:

$$\pi(a_t, s_t) = p(a_t|s_t)$$

Тобто на основі стану або спостереження з MDP, функція приймає умовний розподіл дій $a \in A$, з якого потім випадковим чином генерується дія a_t.

Отже керуючись такою стратегією агент приймає стан чи спостереження від середовища, і з умовного розподілу дій з простору дій він виконує конкретну дію.

У класичних методах навчання з підкріпленням, які апроксимують стратегію, функція зазвичай наближується у вигляді табличної функції. Такий підхід до наближення функції стає неефективним з практичної точки зору, оскільки при великій розмірності простору станів чи простору дій табличне представлення вимагає значних ресурсів пам’яті.

У новітніх розробках у сфері навчання з підкріпленням все більшу популярність набирає підхід на основі побудови нейронних мереж. Таке навчання називається глибоким навчанням з підкріпленням, і у цих алгоритмах функцію стратегії власне і наближують за допомогою самих нейронних мереж, які виступають параметричними моделями-апроксиматорами. В глибокому навчанні з підкріпленням прийнято позначати функцію або у випадку детерміністичної функції. позначає вектор параметрів моделі.

Одже, постановка задачі глибокого навчання з підкріпленням сформулюється наступним чином:

$$p_\pi(\tau) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

І оскільки тепер функція π_θ – параметрична, то можна шукати θ^* замість π_θ^*:

$$\theta^* = argmax_{\pi_\theta}\mathbb{E}_{\tau\sim p_\theta(\tau)}\sum_t R(s_t,a_t)$$

2.3 Кумулятивна винагорода, функція вартості та Q-функція

Функція підкріплення $R(s,a)$ є дуже важливим елементом навчання з підкріпленням, завдяки якій воно і отримало свою назву. Сенс її полягає в тому, що фактично вона представляє собою функцію середньої винагороди, яку отримує агент за виконання дії $a \in A$ в стані $s \in S$.

Але на практиці та в алгоритмах, розглядається не власне сама нагорода за дію у певному стані, а кумулятивна винагорода за певну траєкторію τ [11].

Кумулятивною винагородою за скінчений час (finite-horizon undiscounted return) називають таку винагороду, яку агент отримав загалом протягом скінченої кількості кроків на певній траєкторії.

Запишемо формулу (2.3) для такої винагороди.

$$R(\tau) = \sum_{t=0}^{T} R(s_t, a_t), \tag{2.3}$$

де τ – траєкторія за час T.

Така винагорода справедлива тільки тоді, коли траєкторія є скінченою, тобто епізод є обмеженим за часом. Прикладом такого агенту може бути такий, що збирає деталі маніпулятором за певний період часу. Приклад навчання такого агенту наведено на рисунку 2.6. Також важливо, що для комбінаторних ігор розглядається саме цей вид винагороди, оскільки через умови, покладені на комбінаторні ігри вони завжди мають завершуватися за скінчений час, і тоді одна повна гра може вважатися одним епізодом із певною траєкторією τ.

Прикладом такого навчання є координація рухів маніпулятора (рисунок 2.6):

Рисунок 2.6 – Навчання координації рухів маніпулятора та зображення знятого камерою

Для нескінчених траєкторій, які виникають у тих випадках, коли один епізод не є обмеженим за часом, вводитися поняття дисконтованої кумулятивної винагороди.

Визначимо дисконтовану кумулятивну винагороду за нескінчений час (infinite-horizon discounted return) формулою (2.4), яку агент отримує за нескінчену кількість кроків:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t), \tag{2.4}$$

де $0 < \gamma < 1$ – коефіцієнт дисконтування.

За накладання певних умов на функцію винагород $R(s, a)$, коефіцієнт дисконтування гарантує збіжність ряду (2.4). Однією з головних його функцій є виконання функції ваги для винагороди. Винагороди, які були отримані раніше, матимуть більший вклад до кумулятивної нагороди, аніж подальші за

часом. Також коефіцієнт дисконтування дозволяє “зрозуміти” агенту, що більш привабливими є ті винагороди, які він може отримувати раніше. Важливою властивістю, яку привносить коефіцієнт дисконтування є зменшення значення доданків, а отже він зменшує загальну дисперсію кумулятивної винагороди. Цей ефект важливий для алгоритмів навчання з підкріпленням, які використовують градієнт стратегії для навчання.

Наступними важливими поняттями, які базуються на кумулятивній винагороді є функції вартості та Q-функція.

Функцією вартості називається функція $V^{\pi}(s)$, що означає очікувану кумулятивну нагороду за траєкторію τ при стратегії π, за умови, що поточним (можна звести до початкового) станом є s. Позначимо формулою (2.5) функцію вартості:

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}(R(\tau)|s_0 = s) \tag{2.5}$$

Q-функцією називається $Q^{\pi}(s,a)$, що означає очікувану кумулятивну винагороду за траєкторію τ при стратегії π, за умови, що поточний (початковий) стан – s, і дія агенту – a. Позначимо формулою (2.6) функцію вартості:

$$Q^{\pi}(s,a) = \mathbb{E}(R(\tau)|s_0 = s, a_0 = a) \tag{2.6}$$

Ці функції пов’язані наступним рівнянням:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)]$$

Важливо розглянути оптимальні функції вартості та Q-функцію. Ці функції відповідають оптимальній стратегії.

Оптимальною функцією вартості $V^{*}(s)$ називається функція вартості (2.7), що відповідає оптимальній стратегії π^{*}:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}(R(\tau)|s_0 = s) \tag{2.7}$$

Оптимальною Q-функцією $Q^*(s,a)$ називається Q-функція (2.8), що відповідає оптимальній стратегії π^*:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}(R(\tau)|s_0 = s, a_0 = a) \tag{2.8}$$

Вони пов’язані наступним рівнянням:

$$V^*(s) = \max_{a} Q^*(s,a)$$

Ці функції мають важливу роль у алгоритмах навчання з підкріпленням, часто апроксимуються саме ці функції, а не функція підкріплення.

2.4 Рівняння Беллмана

Приведені у минулому підрозділі функції вартості та Q-функція є важливими концептами у навчанні з підкріпленням. Для їх апроксимації для навчання використовують динамічне програмування [12] (формули (2.9) та (2.10)).

$$J_0(x) = 0, \tag{2.9}$$

$$J_{k+1}(x) = \inf_{u\in U(x)} \mathbb{E}\{g(x,u,w) + J_k[f(x,u,w)]\}, k = 0,..,N-1 \tag{2.10}$$

де математичне сподівання береться по мірі $p(dw|x,u)$.

Рівняння Беллмана, яке є основою ДП, формалізує принцип того, що вартістю поточного стану середовища є очікувана кумулятивна винагорода, яку отримують агенти за знаходження у цьому стані плюс вартість стану в

який вони перейдуть далі. Обидві функція вартості та Q-функція підходять під даний принцип.

Рівняння Беллмана (2.10) для функції вартості (2.6):

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot,S),\, s'\sim P(\cdot|s,\,a)}[R(s,a) + \gamma V^{\pi}(s')],$$

де $P(\cdot\,|s,a)$ – ймовірність потрапляння в наступні стани при умові знаходження у початковому стані $s \in S$ і виборі дії $a \in A$.

Аналогічно вводиться рівняння для Q-функції (2.7):

$$Q^{\pi}(s,a) = \mathbb{E}_{\,s'\sim P(\cdot|s,\,a)}\big[R(s,a) + \gamma \mathbb{E}_{a'\sim\pi(\cdot,s)} Q^{\pi}(s',a')\big],$$

де $P(\cdot\,|s,a)$ – той же самий розподіл, що і для функції вартості.

У випадку, коли кумулятивна винагорода не є дисконтованою (випадок комбінаторних ігор), коефіцієнт дисконтування γ формально можна вважати рівним 1.

Рівняння оптимальності Беллмана вводиться для оптимальної функції вартості та оптимальної Q-функції:

$$V^{*}(s) = \max_{a} \mathbb{E}_{s'\sim P(\cdot|s,\,a)}[R(s,a) + \gamma V^{*}(s')],$$

$$Q^{*}(s,a) = \mathbb{E}_{\,s'\sim P(\cdot|s,\,a)}\left[R(s,a) + \gamma \max_{a'} Q^{*}(s',a')\right],$$

Саме рівняння оптимальності Беллмана для оптимальних функцій використовуються для пошуку оптимальної стратегії π^{*}.

2.5 Value iteration та Policy iteration

Класичними алгоритмами навчання з підкріпленням, які використовують рівняння оптимальності Беллмана є алгоритм Ітерація вартостей (Value iteration) та Ітерація стратегій (Policy Iteration) [11].

Алгоритм Ітерації вартостей полягає в обчислені оптимальної функції ітеративно, покращуючи на кожному кроці оцінку $V(s)$. Суть алгоритму полягає в тому, що спочатку оцінка $V(s)$ ініціалізується випадковим чином. Далі на кожній ітерації алгоритму оновлюється оцінка Q-функції Q(s,a), за якою знаходиться оцінка V(s). Ітерації виконуються до збіжності V(s). Алгоритм ітерації вартостей має гарантовану збіжність до оптимальних значень для марковського процесу прийняття рішень. Наведемо псевдокод алгоритму Ітерації вартостей (рисунок 2.7).

Ітерація вартостей

Довільно ініціалізувати масив V для всіх $s \in S$

Повторювати
$\Delta \leftarrow 0$
Для кожного $s \in S$:
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
до тих пір, поки $\Delta < \varepsilon$

Детерміністична стратегія:
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Рисунок 2.7 – Псевдокод алгоритму Ітерації вартостей

Алгоритм Ітерації стратегій полягає у тому, що оцінюється на функція вартості, а власне сама стратегія і ітерація іде по самим стратегіям. Оскільки в алгоритмі іде оптимізацію власне самої стратегії, то і розв’язок може бути

знайдений раніше ніж у Ітерації вартостей. Алгоритм Ітерації стратегій зображений у вигляді псевдокоду на рисунку 2.8.

Ітерація стратегій

1. Довільно ініціалізувати $V(s) \in \mathbb{R}$ та $\pi(s) \in A(s)$ для всіх $s \in S$

2. Оцінка стратегії:
   Повторювати
   $\Delta \leftarrow 0$
   Для кожного $s \in S$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow max(\Delta, |v - V(s)|)$
   до тих пір, поки $\Delta < \varepsilon$

3. Покращення стратегії:
   policy-stable $\leftarrow True$
   Для кожного $s \in S$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   Якщо old-action $\neq \pi(s)$, тоді policy-stable $\leftarrow False$
   Якщо policy-stable, тоді policy-stable $V \approx v^*$ і $\pi \approx \pi^*$, інакше перейти до кроку 2

Рисунок 2.8 – Псевдокод алгоритму Ітерації стратегій

Перевагою цих основних алгоритмів є те, що обидва ці методи мають властивість гарантованої збіжності для MDP та деяких модифікацій процесу прийняття рішень [13].

Недоліком цих алгоритмів є те, що тут функції вартості та Q-функція представляються у табличному вигляді, а як вже зазначалося, такий підхід може бути неефективним з практичної точки зору реалізації через великий розмір простору станів чи простору дій. Так наприклад цей недолік робить неефективним використання цих алгоритмів у комбінаторних іграх через величезний розмір простору станів та простору дій у них. Іншим суттєвим недоліком алгоритмів є те, що вони потребують визначення оператору переходу станів, тобто вони працюють тільки коли відома модель середовища.

Методи Ітерації вартостей та Ітерації стратегій дають гарне представлення того, як відбувається процес пошуку оптимальних стратегій і вони надихнули на розробку багатьох інших алгоритмів RL.

2.6 Дослідження та експлуатування в навчанні з підкріпленням

Одним із головних випробувань, які постають перед дослідником під час дослідження задачі навчання з підкріпленням є проблема дослідження та експлуатації. Її суть полягає у тому, аби під час навчання правильно збалансувати можливості дослідження агентом простору стратегій із експлуатацією найкращих із них [14,15].

Дослідження – це здатність алгоритму шукати нові стратегії, які будуть кращими аніж за ті, які агент уже знає. Чим більше алгоритм використовує дослідження, тим швидше він може знайти кращі стратегії, і тим швидше він навчить агента відповідній оптимальній стратегії. Проте є проблема, алгоритм може і не знайти більш оптимальних стратегій, а отже час роботи алгоритму буде витрачено на пошук кращої стратегії, але результату досягнуто не буде.

У свою чергу експлуатація – це здатність алгоритму використовувати найкращі стратегії, чи близькі до них, а не намагатися знайти кардинально інші кращі стратегії.

У випадку дослідження бажаною поведінкою алгоритму була б така, яка б забезпечувала певну збіжність алгоритму, тобто щоб кумулятивна винагорода зростала при зміні стратегії. Проблема полягає в тому, аби в який момент часу алгоритм перейшов від дослідження до експлуатації, тобто змусити його перестати шукати нові стратегії з плином часу. Це б дозволило значно зменшити кількість часу витраченого на пошук оптимальної стратегії, оскільки б алгоритми не вираховували можливі інші стратегії, коли відома вже досить гарна стратегія для агенту.

Іншою проблемою дослідження є випадки, які часто зустрічаються у різних іграх, а саме коли винагорода є розрідженою у часі. Це означає, що аби

отримати підкріплення у вигляді винагороди агент має виконати певну довгу стратегічну послідовність дій або відвідати багато станів. Проблема полягає в тому, що алгоритми на початку своєї роботи при дослідженні є досить зашумленими, вони нагадують гаусівський випадковий процес, або ж процес випадкового блукання. У таких випадках алгоритм може пройти велику кількість ітерацій і пройде багато часу, перш ніж цей процес віднайде саме ту стратегію, яка наддасть йому цю винагороду, аби метод почав оптимізацію цієї стратегії.

Зазвичай такі середовища є дуже складними, і алгоритми навчання з підкріпленням мають великі складнощі в подоланні цієї проблеми. Можливим розв'язком цієї проблеми може бути виведення нової функції підкріплення, яка б надавала алгоритму більше інформації щодо його дій, а не лише в кінці одного епізоду. Проте не завжди це можливо чи просто. Прикладом середовища, яке дуже складне для алгоритмів навчання з підкріпленням через вищеописану проблему є середовище гри Montezuma's Revenge (рисунок 2.9).

Рисунок 2.9 – Зображення середовища гри Montezuma's Revenge

У цій грі для перемоги у ній треба пройти велику кількість різних кімнат-рівнів. У кожній кімнаті агент має знайти ключ від двері, власне самі двері, причому не потрапити у пастки на кожному рівні. Всі ці умови призводять до того, що агенту важко виконати завдання, які стоять перед ним, і він не може пройти навіть початкову кімнату, а отже він не отримує підкріплення його дій від алгоритмів. Це і означає, що нагорода у цьому середовищі є розрідженою у часі.

Для дослідження алгоритмів навчання з підкріпленням було запропоновано використовувати певні набори середовищ, на яких би можна було порівнювати їх ефективність. Одним з цих наборів є ігри з ігрової консолі Atari 2600, в який і входить гра Montezuma’s Revenge як одне із найскладніших середовищ. Так, у статті [16] де було запропоновано алгоритм DQN для пошуку стратегій, який буде розглянуто далі у роботі, можна побачити що такі середовища дають дуже незадовільні результати. Порівняння алгоритму DQN з іншим лінійним алгоритмом та професійним гравцем людиною наведено на рисунку 2.10 нижче.

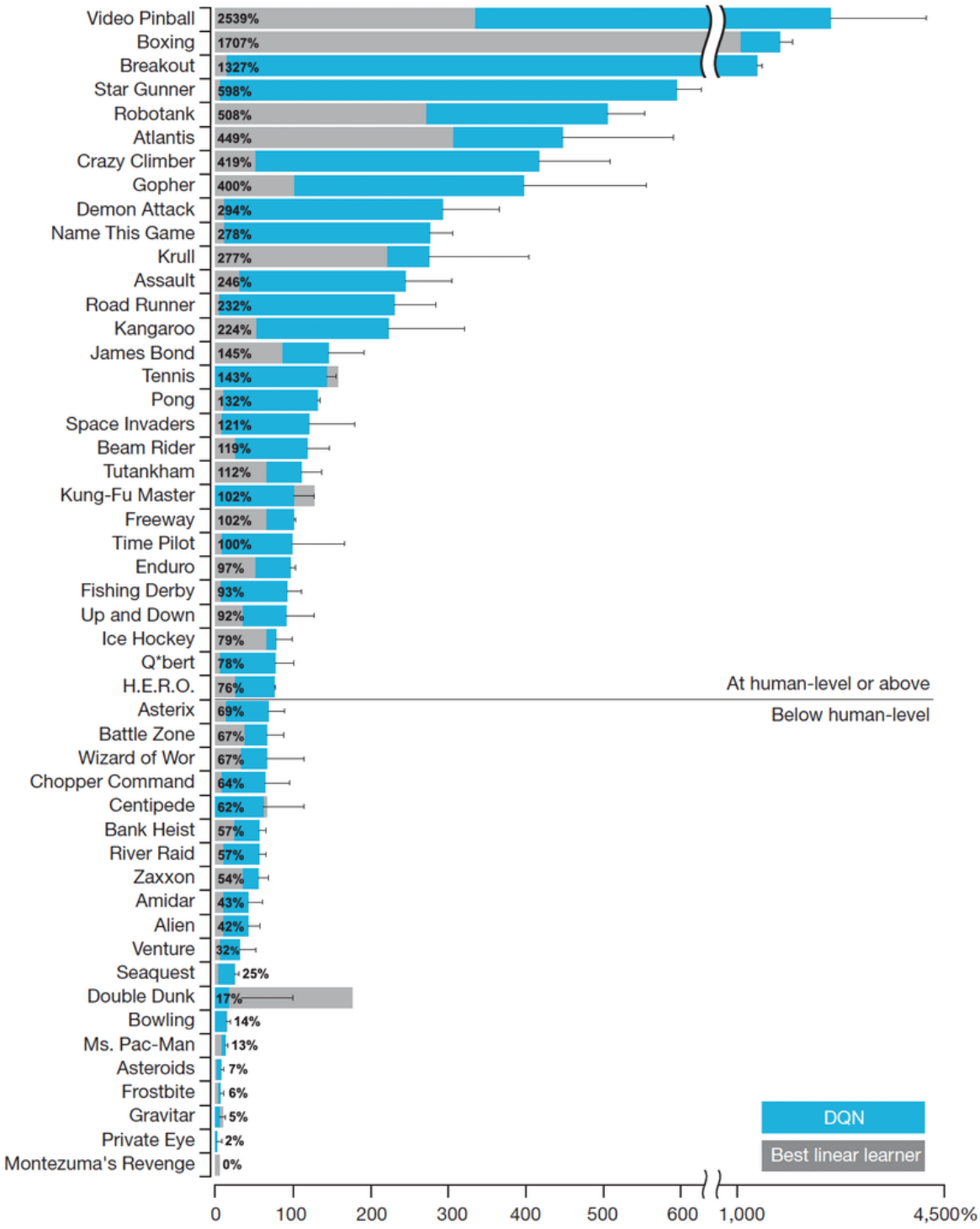


Рисунок 2.10 – Порівняння результатів алгоритму DQN з людиною та іншим алгоритмом навчання, у тому числі у середовищі Montezuma’s Revenge

Вирішення даних проблем для таких середовищ є однією із головних цілей сучасних наукових досліджень. Одним із варіантів подолання цієї проблеми є застосування методу підрахунків [17], ідея якого полягає в тому,

щоб агент отримував більшу винагороду за відвідування тих станів, в яких він був менше ніж в інших. Іншою, досить успішною моделлю, яка ставить перед собою задачу вирішення проблеми дослідження є навчання засноване на допитливості [18]. Також в останній час з'являються все нові методи та підходи, які дозволяють вирішити проблему дослідження навіть для таких складних середовищ, як Montezuma's Revenge [19,20].

Класичним способом запровадження дослідження в алгоритмах навчання є використання так званих ε −жадібних стратегій. Їх суть полягає у тому, що на кожній ітерації алгоритму він використовує найкращу стратегію, слідуючи принципу експлуатації, проте з певною малою ймовірністю ε алгоритм може - це забезпечує певний шанс віднайти кращу стратегію, аніж відома зараз алгоритму.

2.7 Класифікація алгоритмів навчання з підкріпленням

2.7.1 Способи класифікації

Алгоритми навчання з підкріпленням можна класифікувати за різними ознаками. Одним із загальноприйнятих способів є поділ на алгоритми навчання з моделями середовища (model-based) і алгоритми навчання без моделей середовища (model-free).

Такий вид класифікації можна зобразити на схемі (рисунок 2.11):

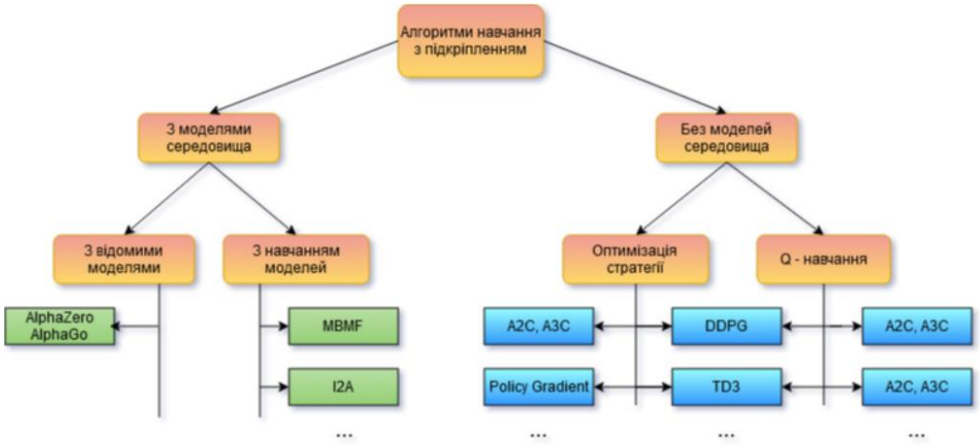


Рисунок 2.11 – Схема класифікації алгоритмів навчання з підкріпленням

Загалом схему роботи алгоритму навчання з підкріпленням можна описати діаграмою на рисунку 2.12:

Рисунок 2.12 – Схема роботи алгоритму навчання з підкріпленням

Загальна структура майже всіх алгоритмів навчання з підкріпленням є такою:

Генерування нових прикладів. На цьому етапі алгоритму виконується генерація послідовностей (s_t, R_t, a_t, s_{t+1}) за допомогою взаємодії агента із середовищем, який діє за певною стратегією. Якщо власне саме середовище не доступне, то використовується симулятор такого середовища.

Навчання додаткових моделей. На цьому етапі відбувається навчання додаткових моделей (модель середовища, моделі, що оцінюють функцію вартості, моделі середнього рівня тощо) на згенерованих прикладах.

Покращення стратегії. На цьому етапі відбувається оновлення стратегії за певним алгоритмом. Не всі алгоритми апроксимують стратегію.

Іншим способом класифікації алгоритмів навчання є поділ на онлайн та не онлайн алгоритми. Онлайн алгоритм навчання з підкріпленням – це метод навчання з підкріпленням, коли на етапі генерування генерується лише один приклад.

Також алгоритми поділяють на on-policy (ті, що використовують поточну стратегію для генерування прикладів) та off-policy (використовують найкращу з відомих стратегій для генерування прикладів). Алгоритми off-policy дають можливість використовувати згенеровані раніше приклади повторно, що пришвидшує їх роботу.

2.7.2 Алгоритми з та без моделей середовища

Одним із видів алгоритмів навчання з підкріпленям є алгорими, що використовують моделі середовища у своїй роботі.

Як можна побачити з формули розподілу траєкторій (2.1), вона складається з двох видів множників - $\pi(a_t|s_t)$, які відповідають за стратегію, і $p(s_{t+1}|s_t, a_t)$, які насправді є розподілом, що відповідає за модель середовища. У алгоритмах з моделями середовища, якщо ми визначимо функції підкріплення, то ми зможемо виразити оптимальні стратегії для середовищ на

основі їх моделі напряму. Навчання з підкріпленням, засноване на моделях середовища, часто спирається на теорію керування. Дії насправді є керуванням певної системи, яка в навчанні з підкріпленням називається середовищем. Алгоритми навчання з підкріпленням, засновані на моделях середовища, іноді використовують підходи з теорії керування, такі як пошук оптимального регулятора, який знаходить оптимальну послідовність:

$$\tau = (x_1, a_1, x_2, u_2, \dots x_T, u_T)$$

Тобто у алгоритмах цього типу, відшукується оптимальна траєкторія з найменшою вартістю, замість того аби максимізувати винагороду для агента.

Ці алгоритми також поділяються на алгоритми з відомою існуючою моделлю, чи ті, де вона вивчається алгоритмом. Прикладом алгоритму, де модель середовища відома є згаданий вище алгоритм AlphaZero. На відміну від свого попередника AlphaGo, AlphaZero отримує лише стан дошки для гри у Ґо в якості спостереження. Це дозволило застосувати алгоритм AlphaZero для багатьох інших ігор, як наприклад шахи, сьогі чи навіть Крестики Нолики.

До алгоритмів, що навчаються середовищу, належать такі алгоритми Моделі Світу (World Models), I2A (Imagination-Augmented Agents) [21], Model-Based RL with Model Free Fine-Tuning (MBMF) [22].

Інший вид алгоритмів, алгоритми без моделі середовища, вони передбачають оцінку оптимальної стратегії напряму, без використання знання середовища. Зазвичай ці алгоритми працюють покроково, покращуючи стратегію на кожній ітерації. Такі алгоритми можна вважати такими, що можна описати словами “метод проб і помилок”. Особливістю цього підходу є те, що власне під час самого алгоритму не можливо сказати яким буде наступний стан середовища.

Прикладом такого підходу є, наприклад, алгоритми оптимізації стратегії та Q-навчання.

2.8 Q – навчання

Одним із найпопулярніших алгоритмів навчання з підкріпленням є алгоритми Q-навчання.

Алгоритми Q-навчання поділюються на 2 типи:

Відома Q-функція дозволяє обирати наступні дії для агента, не маючи моделі для власне самої функції стратегії:

$$\pi'(a_t, s_t) = \begin{cases} 1, & \text{якщо} = argmax_{a_t} Q^{\pi}(s_t, a_t) \\ 0, & \text{інакше} \end{cases}$$

Якщо Q^{π} відома, то дія a_t буде найкращою за будь яку іншу дію $a_t \sim \pi(a_t, s_t)$. Якщо ж $Q^{\pi} = Q^*$, отримана таким чином стратегія буде оптимальною. Тому алгоритм Q-навчання знаходить оцінку Q^{π}, що дозволяє дозволяє знайти оптимальну стратегію π^*, явно не моделюючи її. Цей метод називається табличним Q-навчанням.

Оптимальну Q-функцію можна знайти, мінімізуючи формулу (2.11), також відому як помилку Беллмана:

$$\varepsilon = \frac{1}{2}\mathbb{E}_{a,b\sim\beta}\left[Q_\theta(s,a) - \left[R(s,a) + \gamma \max_{a'} Q_\theta(s',a')\right]\right], \qquad (2.11)$$

де β – розподіл всіх можливих епізодів;

θ – набір параметрів моделі, яка моделює Q-функцію.

Якщо $\varepsilon = 0$, тоді $Q_\theta(s,a) = R(s,a) + \gamma \max_{a'} Q_\theta(s',a')$, що відповідає визначенню Q-функції (2.8).

Наведемо псевдокод цього алгоритму на рисунку 2.13.

Q - навчання

Повторювати

Генеруємо приклади виду $D = (s_i, a_i, s_{i+1}, r_i)$

Повторювати k разів:

$y_i \leftarrow R(s_i, a_i) + \max_{a'} Q_\theta(s'_i, a'_i)$

$\theta \leftarrow \arg\min_\theta \frac{1}{2} \sum_i ||Q_\theta(s_i, a_i) - y_i||^2$

до збіжності

Рисунок 2.13 – Псевдокод Q-навчання

Ці два алгоритми мають суттєву різницю: табличне Q-навчання має гарантовану збіжність до оптимальної стратегії, а другий алгоритм не має.

Другий алгоритм фактично використовує у своїй роботі оператор Беллмана:

$$T(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s,a) + \gamma \max_{a' \in A} Q(x', a')]$$

Цей оператор є стиском за p_∞-нормою. У той же час друга дія на кроці алгоритму є насправді проектуванням на простір можливих моделей, яке також є оператором стиску, тільки за нормою L_2. Композиція цих операторів не є стиском, а отже збіжність не гарантована.

Однак, табличне представлення функцій не є оптимальним з обчислювальної точки зору, тому перший алгоритм на практиці застосовується не дуже часто. Популярність другого алгоритму заснована на швидкості його роботи – цей алгоритм не потребує генерування прикладів за допомогою поточної стратегії, а одже він працює значно швидше. Для того, аби алгоритм Q-навчання збігався часто використовують певні емпіричні прийоми.

2.9 Алгоритм DQN

Алгоритм DQN [16] комбінує навчання з підкріпленням з підходами глибоких нейронних мереж. Нейроні мережі були запропоновані у цьому алгоритмі для того, щоб вони вивчали складні представлення даних, такі як наприклад зображення з екрану ігор, як в наборі середовищ Atari2600, чи щоб краще досліджувати складні та великі простори станів та дій.

Оскільки ціллю агента є обирати дію, які максимізують кумулятивну винагороду, то у цьому алгоритмі ANN використовується для того, аби апроксимувати оптимальну Q-функцію (2.8).

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}(R(\tau)|s_0 = s, a_0 = a),$$

де $R(\tau)$ – дисконтована кумулятивна винагорода визначена формулами (2.3) чи (2.4).

Цей алгоритм глибокого навчання з підкріпленням відрізняється від алгоритмів глибокого навчання з учителем тим, що і вхідні дані, і цільова функція постійно змінюються під час процесу навчання, що призводить до нестабільності навчання.

Оскільки алгоритм залежить від стратегії чи значень функції витрат для того аби генерувати приклади. Проте, все швидко змінюється чим більше ми знаємо що треба досліджувати. Чим більше ми досліджуємо середовище, тим кращі значення наших цільових значень станів та дій. Проте і вхід і вихід моделі можуть збігатися. Отже якщо вдасться достатньо сповільнити зміни в вході та виході моделі, то ми зможемо навчити мережу.

Для того, аби розв’язати ці проблеми, в методі DQN було запропоновано дві ідеї, які дозволяють цьому алгоритму працювати. Першою особливістю алгоритму є механізм названий досвідом відтворення, натхнений біологічними процесами. Його ідея полягає у тому, що велика кількість прикладів генерується і записується у окремий буфер, названий буфером

досвіду. Він потім розбивається випадково на невеликі набори даних, які формують набір даних, стабільний для навчання, на яких потім і навчається ANN. Оскільки алгоритм обирає ці набори даних випадково із буферу досвіду, то данні не залежать один від одного і мають меншу кореляцію один з одним, тим самим більше нагадуючи незалежні, однаково розподілені випадкові величини. Це дозволяє навчати мережу у манері схожої до навчання з учителем.

Другою особливістю алгоритму DQN є те, що у алгоритмі використовується ітеративне оновлення, яке корегує значення Q, що означає що цільові значення оновлюються лише з певною частотою, а отож зменшується кореляція входу і цільової функції. Формально це означає, що для навчання використовується функція втрат, задана наступною формулою:

$$L_i(\theta) = \mathbb{E}_{(s,a,s',r)\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i)\right)^2\right],$$

де θ_i – параметр Q-мережі на ітерації i;

θ_i^- - параметр мережі, що використовується для обрахування цільової функції на ітерації i.

Параметри θ_i^- оновлються параметрами θ_i з Q-мережі лише через певну фіксовану кількість кроків C і залишається незмінними впродовж усього часу між кожними оновленнями. Цю особливість також можна розглядати як дві ANN, одна з яких виправляє значення Q, а інша оновлюється на кожній ітерації алгоритму під час навчання. Тоді ці значення синхронізуються через певну кількість кроків. Ця особливість алгоритму DQN також надає можливість мати стабільний вхідний та вихідний сигнал для навчання і це дає змогу навчати мережу у манері навчання з підкріпленням.

Наведемо оригінальний алгоритм DQN [16] (рисунок 2.14):

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Рисунок 2.14 – Оригінальний алгоритм DQN

Оригінальний алгоритм DQN використовує $\varepsilon -$ жадібну стратегію вибору дії агента. Для покращення навчання на його початку $\varepsilon = 1$, і з часом зменшується до числа від 0.01 до 0.001. Отже під час навчання стратегія для генерації дій має такий вигляд:

$$\pi(a|s) = \begin{cases} 1 + \frac{\varepsilon}{\text{m}} - \varepsilon, & \text{якщо } a^* = arg\max_a Q(s,a) \\ \frac{\varepsilon}{\text{m}}, & \text{інакше} \end{cases},$$

де m – кількість можливих дій.

Такий підхід до вибору стратегії дозволяє представити алгоритму дослідження, яке з плином часу переростає у експлуатацію.

Цей алгоритм є дуже популярним для навчання агентів у різних середовищах. Було розроблено багато модифікацій цього алгоритму, наприклад дуже вдалими модифікаціями є подвійна DQN [23], DQN з

пріоритетним буфером досвіду [24], в якому дані з буферу обираються не рівномірно, Dueling DQN [25] та DQN з Noisy Networks для дослідження [26].

2.10 Алгоритм REINFORCE та Актор-Критик

2.10.1 Градієнт стратегії

Наступним підходом, який використовується у навчанні з підкріпленням є напряму пошук стратегії за допомогою алгоритмів оптимізації. Так, одним із способів пошуку оптимальних стратегій є методи засновані на градієнтному спуску. У навчанні з підкріпленням, метод навчання за допомогою диференціювання очікуваної кумулятивної винагороди називається методами градієнта стратегії [27] (policy gradients).

Запишемо функціонал для максимізації в навчанні з підкріпленням (2.2) через інтеграл:

$$J(\theta) = \int \pi_\theta(\tau) R(\tau) d\tau$$

За правилом логарифмічного диференціювання маємо:

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \pi_\theta \nabla_\theta \log \pi_\theta(\tau)$$

Підставимо цей вираз у функціонал:

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) R(\tau) d\tau$$
$$= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau) R(\tau)]$$

Проте, цього не достатньо, аби обчислити наближення градієнту. У силу рівняння (2.1) ще треба знати динаміку середовища. Запишемо наступне рівняння (2.12):

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \qquad (2.12)$$

Взявши натуральний логарифм з обох сторін цього рівняння, отримаємо:

$$\log \pi_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)$$

Візьмемо градієнт обох частин цього рівняння і запишемо формулою (2.13):

$$\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \left[\log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right] =$$

$$= \nabla_\theta \left[\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) \right] = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \qquad (2.13)$$

Підставляючи (2.13) в (2.12) отримуємо:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=1}^{T} R(s_t, a_t) \right) \right]$$

Застосовуючи незміщену оцінку математичнго сподівання отримуємо формулу (2.14) для незміщеної оцінки градієнта стратегії.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t=1}^{T} R(s_{i,t}, a_{i,t})\right)\right], \quad (2.14)$$

де N – розмір вибірки.

2.10.2 Алгоритм REINFORCE

Формула (X) покладена в основу алгоритму REINFORCE [27]. Наведемо його алгоритм на рисунку 2.15:

REINFORCE

Довільно ініціалізувати параметри θ

Повторювати

Генеруємо приклади за стратегією $\pi_\theta(a_t|s_t)$

$\Delta_\theta J(\theta) \approx \frac{1}{N}\sum_{t=1}^{T}((\sum_{t=1}^{T}\Delta_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^{T} R(s_t, a_t)))$

$\theta \leftarrow \theta + \alpha\Delta_\theta J(\theta)$

Рисунок 2.15 - Псевдокод алгоритму REINFORCE

Отриману оцінку градієнту у певному сенсі можна розглядати як узагальнення градієнту з ММП:

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right]$$

Тобто сам метод максимальної правдоподібності можна отримати з оцінки градієнту стратегії, якщо прийняти, що кожен приклад має однакову вагу у вигляді певної винагороди, а градієнт стратегії – це узагальнення ММП, де кожен приклад має різну вагу.

Алгоритм REINFORCE має суттєвий недолік – велика дисперсія оцінки градієнту, що суттєво сповільнює збіжність алгоритму.

2.10.3 Зниження дисперсії градієнту

Причиною значної дисперсії градієнту є значення винагород, що входять до кумулятивної винагороди. На практиці використовують певні методи, аби знизити розмір цих винагород у алгоритмі.

Принцип причинності – принцип полягає в тому, що стратегія в момент часу t' не може впливати на винагороду в час $t < t'$. Це дозволяє зменшити дисперсію, для цього формула (2.14) приймає такий вигляд:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_{i,t}\middle|s_{i,t}\right)\left(\sum_{t\prime=t}^{T}R\left(s_{i,t\prime},a_{i,t\prime}\right)\right)\right]$$

Цей принцип дозволяє суттєво зменшити дисперсію оцінки градієнта, до того ж, оцінка залишається незмінною [28].

Зменшити дисперсію також напряму зменшуючи ці винагороди відніманням. Значення, що віднімаються від винагород називаються базовими вартостями. Прикладом базової вартості є, наприклад, середня кумулятивна винагорода:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\nabla_\theta \log \pi_\theta(\tau)\,[R(\tau)-b]\right],$$

$$b = \frac{1}{N}\sum_{i=1}^{N}R(\tau)$$

2.10.4 Алгоритм Актор-Критик

Вважатимемо, що ми застосували принцип причинності для зменшення дисперсії градієнту. Тоді розглянемо наступну величину, що входить до формули оцінки градієнту:

$$\sum_{t\prime=t}^{T} R\left(s_{i,t\prime}, a_{i,t\prime}\right)$$

Ця величина є оцінкою очікуваної кумулятивної винагороди, за умови що агент застосує дію $a_{i,t}$, знаходячись в стані $s_{i,t}$. Тобто її можна переписати у вигляді:

$$Q(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[R(s_{t'}, a_{t'})|s_t, a_t]$$

Введемо середню базову вартість у такому вигляді:

$$V(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t, s_t)}[Q(s_t, a_t)]$$

Оцінка градієнту (2.14) в нових позначеннях матиме вигляд (2.15):

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(a_{i,t}\middle|s_{i,t}\right) A^{\pi_\theta}\left(s_{i,t}, a_{i,t}\right)\right],$$
$$A^{\pi_\theta}\left(s_{i,t}, a_{i,t}\right) = Q\left(s_{i,t}, a_{i,t}\right) - V\left(s_{i,t}\right), \qquad (2.15)$$

де величина A^{π_θ} називається перевагою (advantage).

Така оцінка забезпечує нищу дисперсію градієнта. Її особливістю є те, що покращення оцінки переваги A^{π_θ} зменшує дисперсію оцінки градієнту. Можна виділити два способи оцінки переваги.

З формули (2.15) видно, що для оцінки переваги можна оцінювати A^{π}, Q^{π} або V^{π}. Наведені далі методи оцінюють останню функцію. Використаємо оцінку V^{π} для отримання виразу переваги:

$$A^{\pi}(s_t, a_t) \approx R(s_t, a_t) + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

Для оцінки V^{π} в глибокому навчанні з підкріпленням використовуються або окремі нейронні мережі, або ті ж мережі що і наближують власне саму стратегію. Існує два основних методи пошуку оцінки функції V^{π}.

Першим способом оцінки є так звана оцінка Монте-Карло:

$$V^{\pi}(s_t) \approx \sum_{t'=t}^{T} R(s_{t'}, a_{t'})$$

Очевидно, що вона є менш ефективною оцінкою, аніж наведена нижче, оскільки має більшу дисперсію:

$$V^{\pi}(s_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} R(s_{t'}, a_{t'})$$

Проте з практичної точки зору отримання оцінки за другою формулою неможливе, оскільки вона потребує повертання у час t багато разів. У свою чергу оцінка Монте-Карло має гіршу дисперсію, проте усе одно демонструє гарні результати під час проведення навчання. Данні і функціонал похибки для навчання мережі, що наближує V^{π} виглядають так:

$$y_{i,t} = \sum_{t'=t}^{T} R\left(s_{i,t'}, a_{i,t'}\right),$$

$$\{(s_{i,t}, y_{i,t})\}$$

$$L(\theta) = \frac{1}{2}\sum_i \left\| \hat{V}_\theta^\pi(s_i) - y_i \right\|^2$$

Другим метод оцінки V^π є оцінка, відома в літературі як bootstapped estimate (бутстреп оцінка).

$$V^\pi(s_t) \approx R(s_{i,t}, a_{i,t}) + \hat{V}_\theta^\pi(s_{i,t+1})$$

Головна відмінність цієї оцінки – використання минулого значення оцінки для отримання нового значення. В такому разі, приклади і функціонал похибки для навчання мережі, що наближує функцію вартості мають вигляд:

$$y_{i,t} = R(s_{i,t}, a_{i,t}) + \hat{V}_\theta^\pi(s_{i,t+1}),$$

$$\{(s_{i,t}, y_{i,t})\}$$

$$L(\theta) = \frac{1}{2}\sum_i \left\| \hat{V}_\theta^\pi(s_i) - y_i \right\|^2$$

В обох методах значення y_i фіксуються як константи, для забезпечення навчання мережі.

Розглянемо алгоритм Актор-Критик [29][30]. Назва Актор-Критик походить від назви моделей, які наближують навчаються у цьому алгоритмі функції вартості та стратегії. Модель, що наближує стратегію зветься Актором, оскільки вона відповідає за поведінку агента, а модель, що наближує функцію вартості називається Критиком.

Алгоритм Актор-Критик (рисунок 2.16) використовує один із наведених вище методів для пошуку стратегії:

Актор-Критик

Повторювати

Згенерувати траєкторії (s_i, a_i), використовуючи стратегію $\pi_\theta(a|s)$

Оновити оцінку $\widehat{V}_\theta^\pi(s)$

$\widehat{A}^\pi(s_i, a_i) = R(s_i, a_i) + \widehat{V}_\theta^\pi(s_{i+1}) - \widehat{V}_\theta^\pi(s)$

$\Delta_\theta J(\theta) \approx \sum_i \Delta_\theta \log \pi_\theta(a_i, s_i) \widehat{A}^\pi(s_i, a_i)$

$\theta \leftarrow \theta + \alpha \Delta_\theta J(\theta)$

Рисунок 2.16 - Псевдокод алгоритму Актор-Критик

Насправді, алгоритм Актор-Критик є досить широким класом алгоритмів. Так існує багато варіацій їх реалізацій та варіантів роботи. Наприклад існує онлайн версія алгоритму для дисконтованої нагороди, off-policy алгоритми типу Актор-Критик та on-policy алгоритми [31].

Алгоритм типу Актор-Критик має певний недолік у порівняні з алгоритмом DQN – він має більш низьку швидкість роботи. Проте його застосування більш теоретично обґрунтоване, оскільки він заснований на алгоритмі стохастичного градієнтного спуску. Також на практиці часто саме алгоритми цього типу показують дуже гарні результати.

2.11 Висновки до розділу 2

В цьому розділі був приведений детальний розбір теорії навчання з підкріпленням. В рамках цього розділу було надведено теорію, пов'язану з марковськими процесами прийняття рішень, розглянуто основні концерти у цій теорії. Запропоновано використання моделі взаємодії агенту із середовищем та надано її формалізацію у вигляді марковського процесу прийняття рішень.

Надалі було приведено узагальнення MDP на випадок середовища з неповною інформацією. Розглянуто поняття стратегії і її ролі у навчанні з

підкріпленням. На основі цієї теорії будо наведено основну задачу навчання з підкріпленням.

Наступним кроком був більш детальний огляд поняття стратегія та було висвітлено види її апроксимації. Для цього було наведено поняття про Q-функцію та функцію вартості. Було коротко розглянуто метод знаходження оптимальних функцій – динамічне програмування Беллмана. У якості прикладів алгоритмів, що використовують динамічне програмування наведено алгоритми ітерації стратегій та ітерації вартостей.

Дуже ретельно було розглянуто основну проблему навчання з підкріпленням - проблема дослідження та експлуатації. Ця проблема виникає у середовищах, в яких агентам важко отримати винагороду, а би вони почали напрямлений пошук оптимальної стратегії. Саме до такого виду середовищ відносяться середовища комбінаторних ігор, оскільки агентам потрібно зробити багато дій впродовж однієї партії, а нагороду за весь цей епізод вони отримують лише у її кінці. Пропонуються методики вирішення проблеми дослідення.

Також була розглянута класифікація алгоритмів навчання з підкріпленням, розглянуто основні етапи алгоритму як генерування прикладів, навчання моделі середовища чи оновлення значень функції, що апроксимується та оновлення стратегії. Для різних видів алгоритмів наведено приклади.

Далі розглядаються алгоритми, пов'язані з дослідженням цієї роботи. Наводиться алгоритм Q-навчання, його варіації. Приводяться його недоліки та переваги. Q-навчання є основою до наступного розглянутого методу – алгоритм DQN, що комбінує методи глибокого навчання нейронних мереж із Q-навчанням. Приведено особливості його роботи а також найдієвіші модифікації алгоритму.

Далі у розділі було розглянуто алгоритми, що використовують градієнт стратегії для навчання. Такими алгоритмами є алгоритм REINFORCE та Актор-Критик. Для приведення цих методів було приведено виведення

незміщеної оцінки градієнту стратегії. Розглянуто проблему великої дисперсії оцінки градієнта стратегії та наведені методи її зниження.

Далі розглянуто особливості алгоритму Актор-Критик та його недоліки і переваги.

Теорія, розглянута у цьому розділі використовується у цій роботі для навчання агента, здатного грати у середовищі комбінаторної гри.

Розділ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОБУДОВИ МОДЕЛЕЙ ОЦІНОК ФІНАНСОВОГО РИЗИКУ

3.1 Обґрунтування вибору платформи та мови реалізації програмного продукту

У даної роботі розглядається задача навчання агенту для гри у комбінаторні ігри. Для розробки програми, яка дозволяє навчати агентів за допомогою навчання з підкріпленням була обрана мова програмування Python. Python - це сучасна інтепретуєма мова програмування загального призначення. Назва мови походить від назви британської комедійної групи та однойменного шоу “Монті Пайтон”. Python був започаткований та розроблений як мова програмування, орієнтована на швидкість розробки програмного продукту та високої читаності його коду для зручності розробки проєктів будь-якого масштабу, від невеликих особистих проєктів до продуктів корпоративного рівня.

Особливістю Python є використання динамічної типізації , підрахунку посилань і «збирача сміття», який допомагає ефективно керувати пам'яттю. Python є підтримує багато парадигм програмування, основною з яких є об'єктно-орієнтовна парадигма, проте є підтримка також функціональної, процедурної, аспектно-орієнтовної парадигми тощо. Python має велику стандартну бібліотеку з безліччю зручних інструментів для розробки у ній. Код мови Python виконується у інтерпретаторі Python, який перетворює його на байт-код і виконує на віртуальній машині Python.

Python підтримує високу розширюваність. Так існує безліч додаткових пакетів та бібліотек, які дуже сильно розширюють можливості для розробки програмного забезпечення. Для отримання цих пакетів зазвичай використовуються менеджери пакетів такі як pip чи conda. Вони надають зручний інтерфейс отримання будь-якої з доступних версій бібліотек та контроль за їх використанням та збереженням.

Офіційний репозиторій Python Package Index (PyPI) для стороннього програмного забезпечення Python містить понад 130 000 пакетів для різних задач, які надають безліч функціональних можливостей, наприклад: графічні користувацькі інтерфейси, робота з Веб, робота з даними, автоматизація та тестування, машинне навчання, наукові обчислення, обробка тексту, зображень, звуку, системне програмування тощо.

Python є однією із найпопулярніших мов програмування у світі. До початку 2020 року була підтримка двох офіційних версій мови: Python 2.x і Python 3.x. Більшість функціоналу, створеного на Python 2 було перероблено під версію на Python 3. Нова версія була розроблена, аби виправити деякі недоліки в попередньої версії.

У мові є підтримка багатьох базових типів даних, наприклад числа (цілі числа, з плаваючою точкою, комплексні), рядки та символи(ASCII, Unicode), списки і словники, кортежі тощо. Python є безкоштовним і поширюється як програмне забезпечення з відкритим початковим кодом. Python є це кроссплатформеною мовою, що дозволяє переносити додатки з однієї операційної системи на іншу.

Python в останні роки став основною мовою для розробки програмного забезпечення у сфері машинного навчання та науки про дані. Існує багато бібліотек та фреймворків для розробки програмного забезпечення пов'язаного з машинним навчанням. Саме всі ці фактори стали вируючими при виборі цієї мови для використання у даній роботі. для вирішення задачі побудови агенту-гравця у комбінаторні ігри.

Робота виконувалася у середовищі для розробки мовою Python під назвою Jupyter Lab на платформі Microsoft Windows 10 а також у хмарному середовищі Google Colaboratory. Також епізодично використовувалася платформа на основі серверної операційної системи Linux Ubuntu 18.04 у сервісі хмарних обчислень AWS.

3.2 Вимоги для програмного продукту

Програмний продукт має надавати можливість користувачу обрати певну комбінаторну гру , для якої він хоче отримати агента. Важливим є можливість зміни параметрів гри, як наприклад розмір дошки для отримання різних результатів дослідження і можливого спрощення чи укладення середовища. Далі відбувається навчання агенту у заданому користувачем середовищі і збереження отриманого агенту. Має бути передбачена можливість тестування результатів навчання на епізодах, що не беруть участь у навчанні а також можливість гри користувача проти навченого агента для більш ретельної перевірки його. Можливе використання певного інтерфейсу для взаємодії з користувачем.

Зобразимо схему роботи продукту (рисунок 3.1):

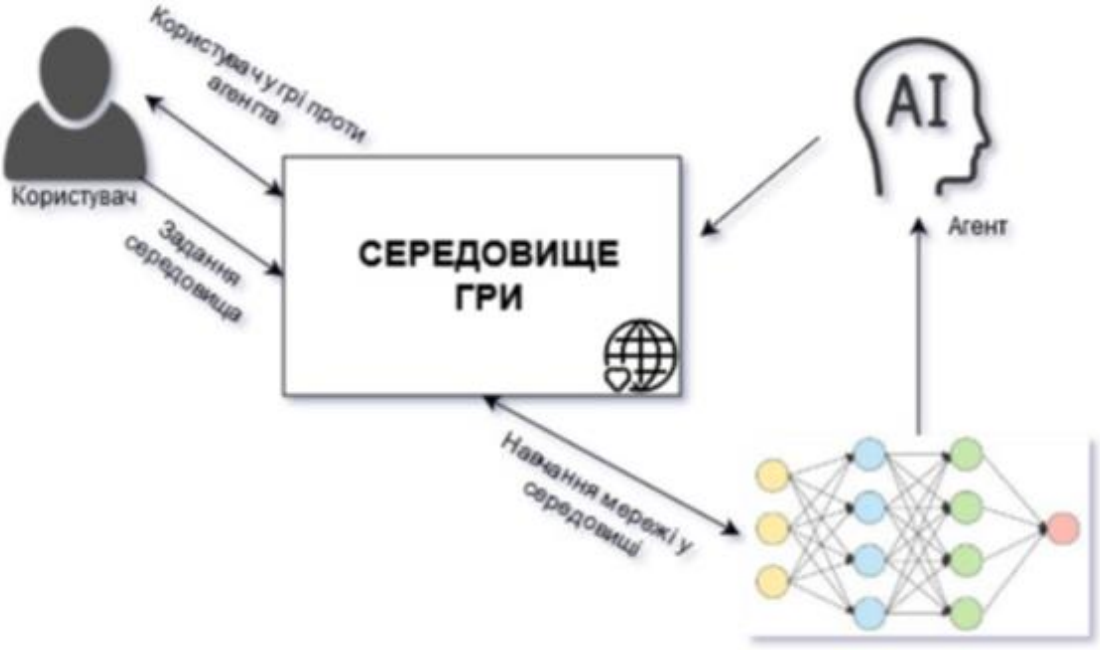


Рисунок 3.1 – Схема роботи програмного продукту

3.3 Середовище для навчання

3.3.1 Фреймворк OpenSpiel

У 2019 році, компанія DeepMind, яка займається дослідженням та розробкою штучного інтелекту та технологій машинного навчання випустила фреймворк під назвою OpenSpiel [32], створений для дослідження загального навчання з підкріпленням пошуку / планування в іграх. OpenSpiel підтримує ігри n-гравців (одноагентні, та багатоагентні ігри) з нульовою сумою, кооперативні ігри та ігри з загальною сумою, послідовними та одночасними, строго поворотом та одночасним переміщенням з повною та неповною інформацією, а також традиційними багатоагентними середовищами такі як (частково і повністю спостережувані) моделі світу у вигляді сітки та соціальні дилеми. OpenSpiel також включає інструменти для аналізу динаміки навчання та інших загальних показників оцінювання.

Наведемо список деяких з реалізованих у цьому фреймворку середовищ (рисунок 3.2):

Game	Reference	Status
Backgammon	Wikipedia	●
Breakthrough	Wikipedia	●
Bridge	Wikipedia	●
(Uncontested) Bridge bidding	Wikipedia	●
Catch	Mnih et al. 2014, Recurrent Models of Visual Attention, Osband et al '19, Behaviour Suite for Reinforcement Learning, Appendix A	~
Cliff Walking	Sutton et al. '18, page 132	~
Coin Game	Raileanu et al. '18, Modeling Others using Oneself in Multi-Agent Reinforcement Learning	~
Connect Four	Wikipedia	●
Cooperative Box-Pushing	Seuken & Zilberstein '12, Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs	~
Chess	Wikipedia	●
Deep Sea	Osband et al. '17, Deep Exploration via Randomized Value Functions	~
First-price Sealed-bid Auction	Wikipedia	●
Gin Rummy	Wikipedia	●
Go	Wikipedia	●
Goofspiel	Wikipedia	●
Hanabi (via Hanabi Learning Environment)	Wikipedia and Bard et al. '19, The Hanabi Challenge: A New Frontier for AI Research	●
Havannah	Wikipedia	●
Hex	Wikipedia	~
Kuhn poker	Wikipedia	●

Рисунок 3.2 – Деякі з реалізованих ігор у фреймворку OpenSpiel

Серед усіх цих ігор є також і комбінаторні ігри: Прорив, Y, З'єднай Чотири, шахи, Го, Гекс та інші. Список реалізованих середовищ постійно оновлюється та додаються нові середовища.

3.3.2 Середовище гра “Прорив”

Для реалізації в роботі і демонстрації застосування алгоритмів було обрано середовище гри “Прорив”.

Прорив (Breakthrough) — стратегічна настільна гра, яку придумав Ден Тройка (Dan Troyka) у 2000 році. Вона грається на дошці, аналогічній гір у шахи та шашки. У гру грають двоє гравців, немає жодного впливу

випадковостей під час гри, гравці спостерігають повністю стан усієї гри. У цій грі ходи робляться один за одним гравцями по черзі. У грі обов'язково є переможець, тобто ця гра дійсно є комбінаторною грою.

Гра починається з позиції, зображеної на рисунку 3.3:

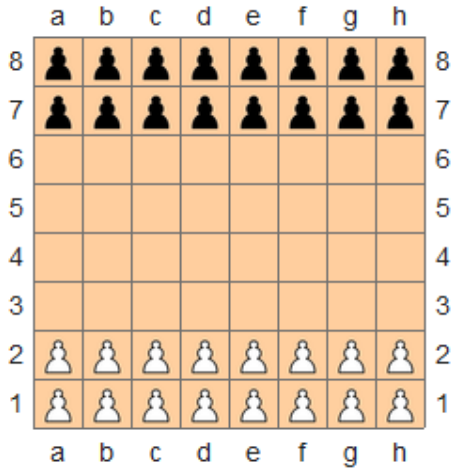


Рисунок 3.3 – початкова позиція у грі Прорив

Існують варіації гри, де дошка має інший розмір, вона не обов’язково має бути квадратною. На кожному кроці гравець зобов'язаний перемістити свою фігуру.

Можливими ходами гравців є переміщення фігури прямо уперед або по діагоналі вперед на 1 клітину, якщо вона порожня. Фігура може піти на клітину, в якій стоїть фігура опонента лише у випадку, коли ця клітина знаходиться лише по діагоналі попереду. Фігура опонента, на місце якої ставиться фігура, що ходить, видаляється до кінця гри із дошки. Дії, які виконують захоплення фігури не є обов'язковими і фігури не можуть ходити декілька разів, якщо це виконання захоплення, на відміну від шашок.

Зобразимо на схемі можливі ходи на прикладі (рисунок 3.4).

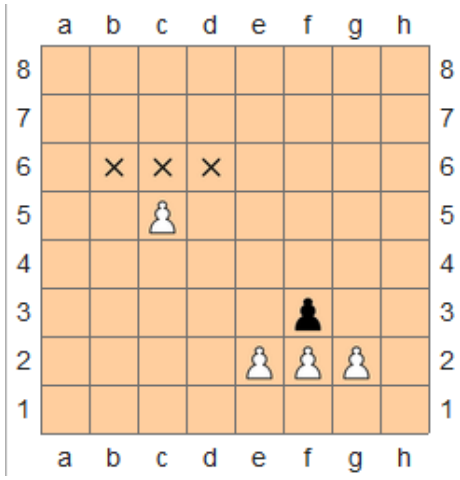


Рисунок 3.4 – Схема можливих ходів

Фігура на c5 може виконати хід на b6, c6, d6. Фігура на f2 не може виконати захоплення чорної фігури, а фігури з e2 та g2 можуть виконати захоплення фігури на f3.

Гарантування лише одного переможця забезпечено тим, що всі фігури на дошці можуть робити крок тільки у напрямку лінії опонента або бути видаленими а також будь-яка фігура має хоча б один допустимий хід по діагоналі. Якщо фігура якогось з гравців досягла останньої лінії на дошці, вважається що наступний гравець більше не має доступних ходів, а одже він програє.

Не зважаючи на прості формулювання правил, це середовище має дуже складну та різноманітну виграшну стратегію, а одже є гарним прикладом для демонстрації можливості застосування методів навчання з підкріпленням.

3.4 Структура реалзації програми

Для реалізації програми була використана версія мови програмування Python 3.6. Ця версія мови програмування є стабільною версією, яку підтримує більшість з наявних взагалі бібліотек для мови Python. Для моделювання середовища та алгоритму було використано фреймворк OpenSpiel, описаний вище. Його альтернативою міг би виступити відомий фреймворк OpenAi Gym,

проте він є більш спеціалізованим на одноагентні середовища, що не є оптимальним для розробки агенту в середовище двох агентів, а OpenSpiel є спеціально розробленим саме для подібних досліджень.

В якості фреймворку для навчання глибоких нейронних мереж було використано фреймворк від компанії Google – TensorFlow. Він використовує механізм диференціальних графів для навчання моделей нейроних мереж. Цей фреймворк є найпопулярнішим вибором у світі для навчання ANN на сьогодення.

Для прискорення навчання було використано графічні прискорювачі типу nVidia Tesla P4, які дозволяють значно прискорити навчання нейронних мереж. Прискорювачі GPU надаються середовищем хмарної розробки Google Colabarotory.

3.5 Результати роботи

Для побудови агенту було обрано метод навчання DQN, архітектурою мережі в алгоритмі обрано багатошаровий персептрон з двома прихованими шарами розміром 64 кожен. Функція активації ReLU (Rectifier Linear Unit) на кожному шарі:

$$f(x) = \begin{cases} 0, & \text{якщо } x \leq 0 \\ x, & \text{якщо } x > 0 \end{cases}$$

На вхід до мережі подається повне спостереження стану середовища, тобто розмір вхідного шару становить кількість клітин на дошці. Вектор спостереження повністю задає стан середовища гри прорив. Вихідний шар мережі має стільки ж нейронів, скільки і можливих дій у середовищі. Результатом роботи мережі є вектор з ймовірностями виконання певної дії. Цей вектор відповідає розподілу випадкових величин, який використовується стратегією $\pi^{\theta}(a|s)$.

Функцією винагороди у цьому середовищі є такою:

$$\mathrm{R}(s,a)=\begin{cases}1, & \text{якщо стан } s \text{ термінальний і а} - \text{переможна дія}\\ 0, & \text{якщо стан } s \text{ не термінальний}\\ -1, & \text{якщо стан } s \text{ термінальний і гравець не може обрати а}\end{cases}$$

Навчання відбувається за схемою, в якій дві мережі DQN однакової архітектури навчаються грі один проти одного. Це дозволило застосувати алгоритм DQN, який є алгоритмом одноагентного навчання з підкріпленням до задачі багатоагентного навчання. У цьому підході агенти є незалежними, тому формально можна вважати, що під час кроку навчання кожного з агентів, інший агент є частиною середовища. Алгоритм виконує 100000 ітерацій навчання, синхронізація значень оцінки $Q^*(s,a)$ відбувається кожні 1000 ітерацій. Для того, аби подолати проблему дослідження використовується $\varepsilon-$ стратегія, надана у розділі де описаний алгоритм DQN.

Для перевірки стратегії агенту використовується підхід гри 1000 епізодів проти агентів, що діють за стратегією $a\sim U(A)$. Цей підхід дозволяє перевірити успішність вивченої стратегії проти будь-яких інших стратегій гравців.

Розглянута спрощена версія гри з дошкою 4 на 4 для порівняння з більш складними середовищами. Результати навчання приведемо на графіку середньої винагороди з часом навчання, також відомі як криві навчання (рисунок 3.5):

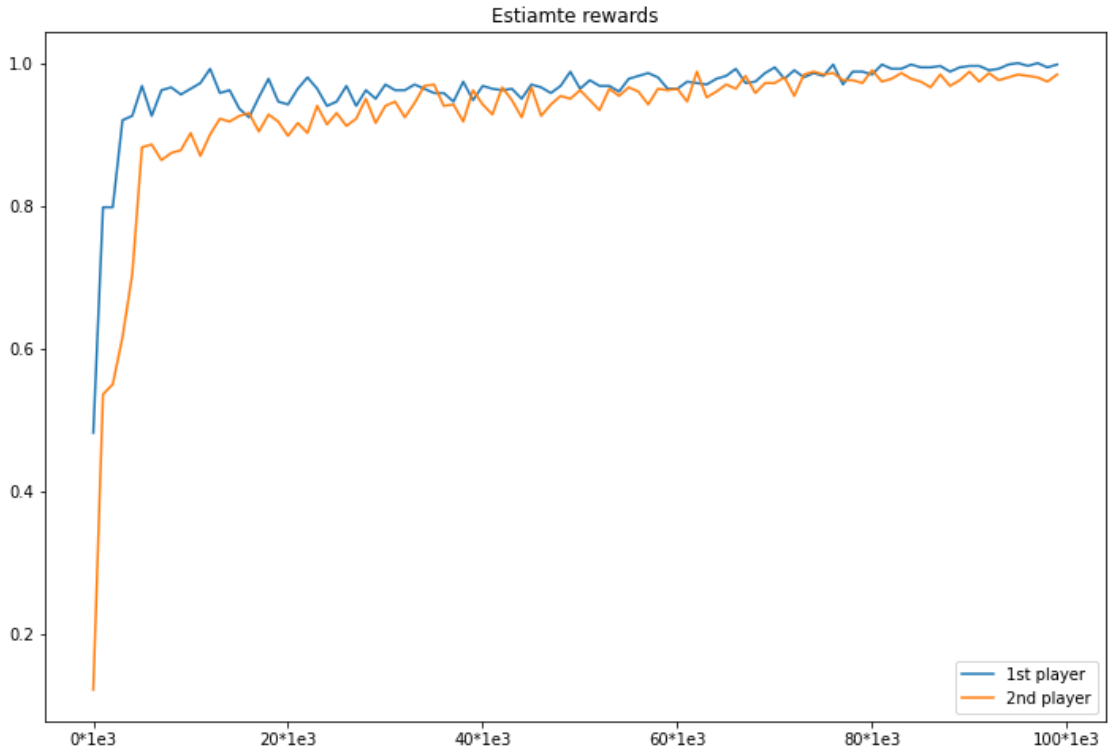


Рисунок 3.5 – криві навчання для гри 4 на 4

Наступні результати для гри на дошці 5 на 10 (рисунок 3.6):

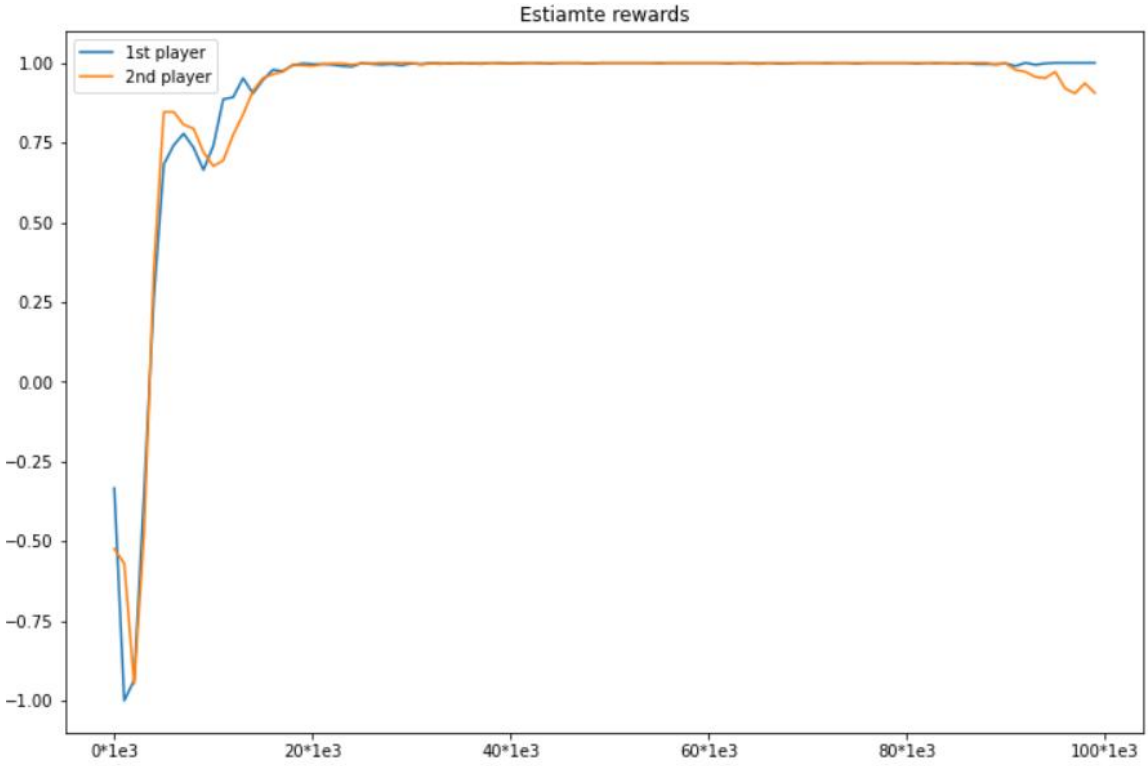


Рисунок 3.6 – криві навчання для 5 на 10 гри

Бачимо, що на початку навчання алгоритм не міг отримати гарні результати. Це пов'язано з проблемою дослідження, бо алгоритм не міг віднайти потрібну виграшну стратегію через розрідженість функції винагород.

Результати для класичної постановки гри з дошкою 8 на 8 і двома рядами фігур (рисунок 3.7):

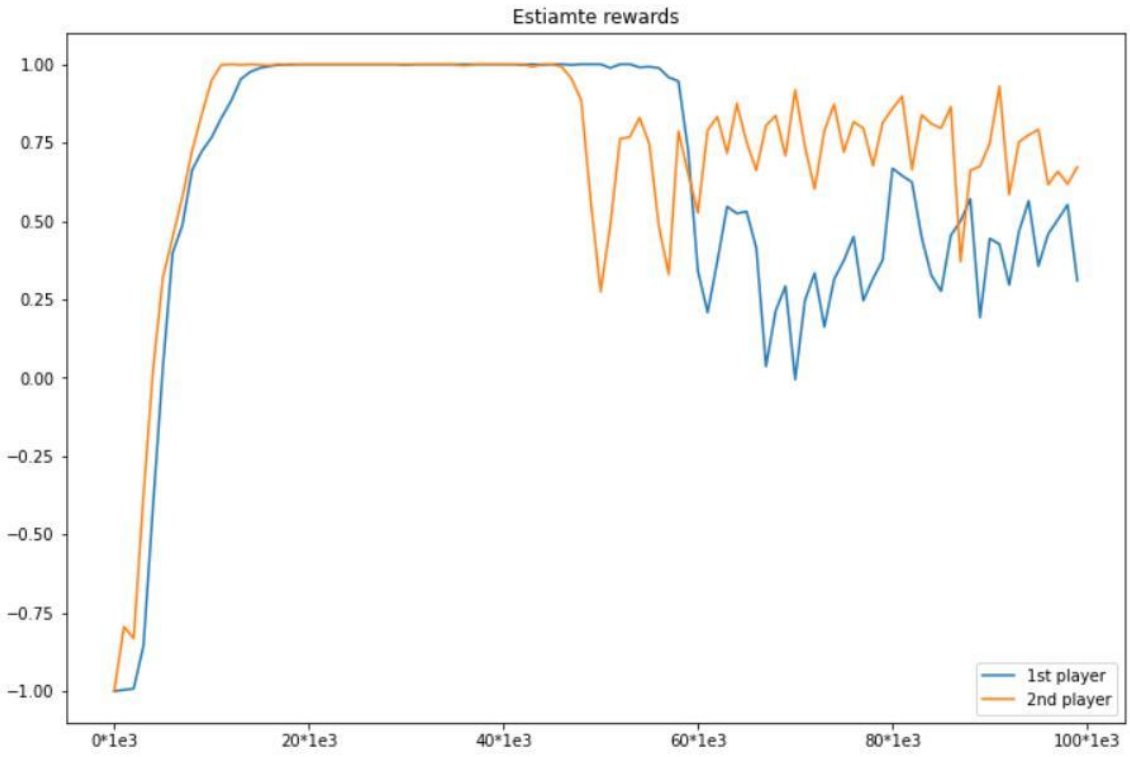


Рисунок 3.7 - Криві навчання агентів для гри 8 на 8

Бачимо, що у даному випадку агенти дуже швидко віднайшли виграшні стратегії, і навчилися загальним стратегіям перемоги за відносно короткий час. Проте після 50000 ітерацій алгоритму агенти почали втрачати можливість перемагати проти узагальненого суперника. Цей ефект пов'язаний з тим, що нейронні мережі почали перенавчантися на стратегії другого гравця, аби перемагати його.

Можливо, використання інших алгоритмів може дати більш кращі результати. Так можна застосувати алгоритм Актор-Критик на меншій кількості ітерацій. Також можливе використання методу DQN з пріоритетним

буфером досвіду або інші модифікації DQN, аби пришвидшити навчання для середовищ з більш складною структурою.

3.6 Висновки до розділу 3

В цьому розділі було розглянуто рішення, задачі пошуку виграшної стратегії для комбінаторних ігор двох гравців. Описано використані технології, обрана мова програмування та фреймворки. Також описано використаний фреймворк для навчання з підкріпленням в іграх OpenSpiel та приклад середовища із нього гра Прорив.

Далі були наведені результати дослідження задачі пошуку виграшної стратегії для комбінаторних ігор двох гравців на прикладі гри Прорив методом DQN. Приведені графіки кривих навчання із їх порівняльним аналізом для середовищ різного характеру.

Висловлено припущення щодо поведінки алгоритмів на даних середовищах та запропоновано можливі інші підходи до розв'язку задачі.

Розділ 4 ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ ПОБУДОВИ АГНТІВ-ГРАВЦІВ КОМБІНАТОРНИХ ІГОР

4.1 Постановка задачі

Проводиться оцінка основних характеристик теорії і отриманих результатів та їх собівартість. Для отримання результатів використовувалась мова програмування Python. Середовище розробки - Jupyter Lab. Робота програми не залежить від технологій реалізації апаратного забезпечення та операційної системи. Нижче приведено аналіз різних підходів до навчання агентів гравців для комбінаторних ігор.

4.2 Обґрунтування функцій програмного продукту

З наявних цілей програмного продукту можна виділити такі основні функції: F_1 – вибір методу навчання агентів для комбінаторних ігор; F_2 – вибір середовища гри; F_3 – вибір мови програмування для реалізації.

Функція F_1: а) класичний підхід навчання з підкріпленням; б) підхід нейронних мереж до навчання з підкріпленням.

Функція F_2: а) гра Ґо; б) гра Шахи; в) сучасні комбінаторні настільні ігри.

Функція F_3: а) Вибір мови програмування Python; б) Вибір мови програмування C++; в) Вибір мови програмування R.

Зображення варіантів комбінацій основних функцій (рисунок 4.1).

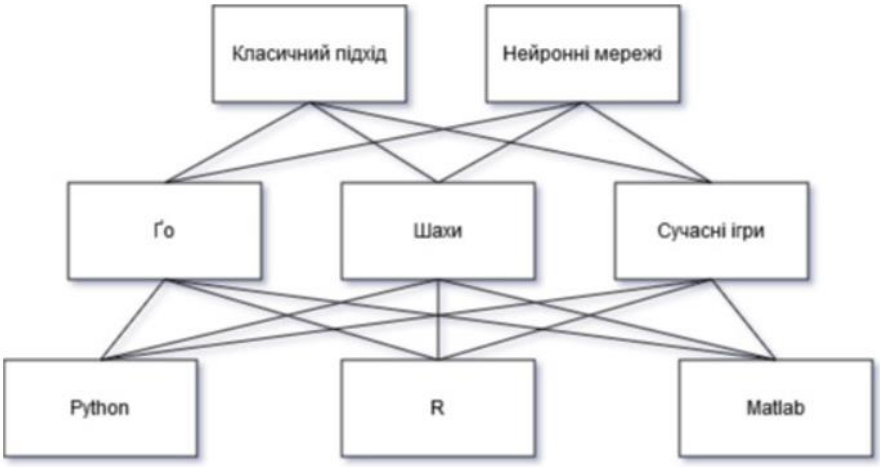


Рисунок 4.1 Морфологічна карта

За допомогою морфологічної карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

Таблиця 4.1 - Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F_1	А	Ґрунтовне теоретичне забезпечення	Неймовірно велика обчислювальна складність для деяких ігор
	Б	Можливість навчання агентів для дуже складних середовищ. Багато бібліотек та фреймворків для роботи.	Велика обчислювальна складність, необхідність у використанні значних ресурсів для навчання.
F_2	А	Наявні сучасні розробки, та відомі методи для розробки агентів, правила відомі і прості для реалізації.	Дуже великий простір станів, висока складність, потрібні потужні ресурси.
	Б	Наявні сучасні розробки, та відомі методи для розробки агентів, правила відомі і прості для реалізації.	Досить великий простір станів, досить висока складність, потрібні потужні ресурси.
	В	Більшість мають відносно невелику кількість комбінацій, перспективний напрямок	Деякі мають великий простір станів, правила можуть бути складними і важкими для реалізації.
F_3	А	Багато готових фреймворків, велика спільнота, простий синтаксис, висока швидкість розробки	Низька швидкість

	Б	Досить висока швидкість, досить велика спільнота	Більша складність коду, часто потребує написання власних методів для алгоритмів
	В	Низька складність, гарна документація	Застаріла технологія, низька швидкість, висока ціна

Тепер, за наявності позитивно-негативної матриці можна робити висновки щодо доцільності використання одних варіантів та недоцільності використання інших. На основі порівняльного аналізу варіантів реалізації основних функцій по їх перевагам та недолікам можна виключити варіанти F_2 а і б , F_3 б і в, тоді варіанти, які залишилися:

$$\mathrm{F}_1\ \text{а}) - \mathrm{F}_2\ \text{в}) - \mathrm{F}_3 - \text{а})$$
$$\mathrm{F}_1\ \text{б}) - \mathrm{F}_2\ \text{в}) - \mathrm{F}_3 - \text{а})$$

Для оцінювання описаних функцій запропонована система параметрів. Опишемо цю систему.

4.3 Обґрунтування системи параметрів програмного продукту

Для характеризації досліджень запропонуємо наступні параметри. Встановимо взаємозв'язок параметрів і функцій.

Для F_1: X1 —швидкодія розробки стратегії (швидкість виконання опереацій); X2 — складність алгоритмізації підходу (Час розробки та написання алгоритму для програми).

Для F_2: X3 — Складність застосування методів розв'язку (Тривалості роботи методів); X4 — Час на реалізацію правил (Час на створення імітації гри).

Для F_3: X5 — Просторова складність робочих модулів (Кількість пам'яті для роботи програми); X6 — Часова складність робочих модулів (Тривалість роботи програми).

4.4 Кількісна оцінка параметрів

Гірші, середні та кращі показники параметрів вибираються на основі вимог замовника та умов експлуатації програмного продукту (Таблиця 4.2).

Таблиця 4.2 – Основні параметри дослідження

Умовні позначення	Одиниці виміру	Значення параметрів		
		гірші	середні	кращі
X1	Оп./с.	2000	8000	12000
X2	Год.	12	6	4
X3	Год.	12	8	3
X4	Год.	20	12	8
X5	Мб.	32	20	12
X6	Хв.	6000	3000	900

На основі таблиці 4.2 будуються графічні характеристики параметрів (рисунок 4.2).

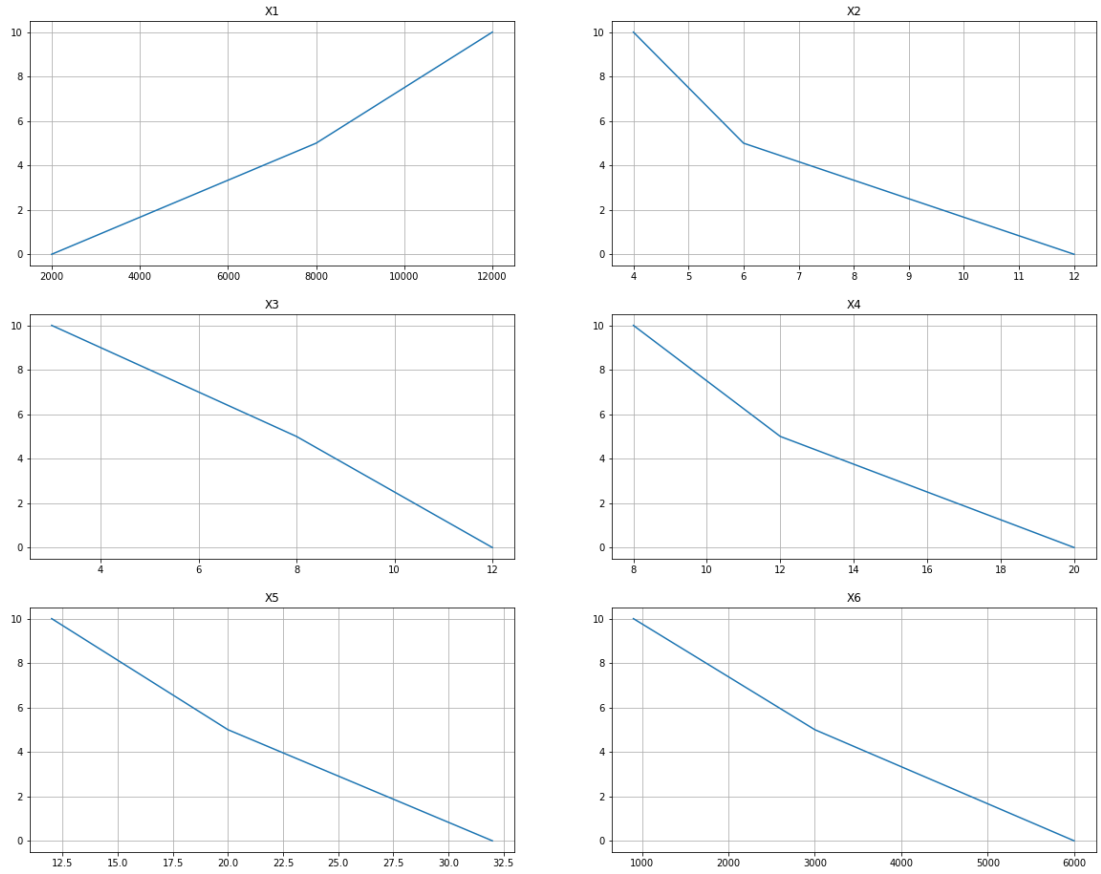


Рисунок 4.2 Графічні характеристики параметрів X1 – X6

4.5 Кількісна оцінка параметрів

На основі ґрунтовного обговорення й аналізу група експертів визначає значимість кожного параметра для конкретно поставленої цілі – розробка програмного продукту.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати експертного ранжування параметрів

Позначення параметра	**Одиниці виміру**	**Ранг параметра за оцінкою експерта**							**Сума рангів R_i**	**Відхилення Δ_i**	**Δ_i^2**
		1	**2**	**3**	**4**	**5**	**6**	**7**			
X1	Год	4	4	6	5	6	6	6	37	12.5	156.25
X2	Год.	1	1	2	3	2	1	1	11	-13.5	182.25
X3	Год.	3	2	1	1	3	4	2	16	-8.5	72.25
X4	Год.	6	5	5	6	5	5	4	36	11.5	132.25

X5	Мб.	5	6	4	4	4	3	5	31	6.5	42.25
X6	Хв.	2	3	3	2	1	2	3	16	-8.5	72.25
Разом		21	21	21	21	21	21	21	147	0	657.5

Обчислимо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 657.5}{7^2(6^3 - 6)} \approx 0.77 > 0.67$$

Експертне ранжування достовірне, виходячи з отриманої нерівності.

Проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4. Зазначимо, що за найбільший ранг приймаємо 1, найменший - 6.

Таблиця 4.4 Попарне порівняння параметрів

Параметри	**Експерти**							**Кінцева оцінка**	**Числове значення**
	1	**2**	**3**	**4**	**5**	**6**	**7**		
X1 та X2	<	<	<	<	<	<	<	<	0.5
X1 та X3	<	<	<	<	<	<	<	<	0.5
X1 та X4	>	>	<	>	<	<	<	<	0.5
X1 та X5	>	>	<	<	<	<	<	<	0.5
X1 та X6	<	<	<	<	<	<	<	<	0.5
X2 та X3	>	>	<	<	>	>	>	>	1.5
X2 та X4	>	>	>	>	>	>	>	>	1.5
X2 та X5	>	>	>	>	>	>	>	>	1.5
X2 та X6	>	>	>	<	<	>	>	>	1.5
X3 та X4	>	>	>	>	>	>	>	>	1.5
X3 та X5	>	>	>	>	>	<	>	>	1.5
X3 та X6	<	>	>	>	<	<	>	>	1.5
X4 та X5	<	>	<	<	<	<	>	<	0.5
X4 та X6	<	<	<	<	<	<	<	<	0.5
X5 та X6	<	<	<	<	<	<	<	<	0.5

4.6 Аналіз рівня якості варіантів реалізації функцій

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступною формулою:

$$K_{\text{ві}} = \frac{b_i}{\sum_{j=1}^{n} b_j} \ ,$$

де $b_i = \sum_{j=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за формулою:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{j=1}^{n} b_j'} \ ,$$

де $b_i' = \sum_{j=1}^{N} a_{ij} b_j$.

Розрахунок вагомості внесемо до таблиці 4.5.

Таблиця 4.5 – Розрахунок вагомості параметрів

Xi	**Xj**						**Перша ітерація**		**Друга ітерація**		**Третя ітерація**	
							$\boldsymbol{b_i}$	$\boldsymbol{K_{\text{ві}}}$	$\boldsymbol{b_i^1}$	$\boldsymbol{K_{\text{ві}}^1}$	$\boldsymbol{b_i^2}$	$\boldsymbol{K_{\text{ві}}^2}$
X1	1.0	0.5	0.5	0.5	0.5	0.5	3.5	0.0972	19.75	0.09945	109.125	0.1005
X2	1.5	1.0	1.5	1.5	1.5	1.5	8.5	0.2361	49.75	0.2506	272.875	0.2513
X3	1.5	0.5	1.0	1.5	1.5	1.5	7.5	0.2083	41.75	0.2103	227.125	0.2091
X4	1.5	0.5	0.5	1.0	0.5	0.5	4.5	0.125	23.75	0.1196	130.875	0.1205
X5	1.5	0.5	0.5	1.5	1.0	0.5	5.5	0.1528	28.75	0.1448	157.125	0.1447
X6	1.5	0.5	0.5	1.5	1.5	1.0	6.5	0.1806	34.75	0.1751	188.875	0.1739
Всього							36	1	198.5	1	1086	1

4.7 Аналіз рівня якості варіантів реалізації функцій

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного продукту розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}$$

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій програмного продукту

Основні функції	Варіант реалізації	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коеф. вагомості параметра	Коеф. рівня якості
F_1	а)	X1	8000	5	0.1005	0.5025
		X2	6	5	0.2513	1.2565
	б)	X1	10000	8	0.1005	0.804
		X2	8	4	0.2513	1.0052
F_2	в)	X3	5	7	0.2091	1.4637
		X4	9	9	0.1205	1.0845
F_3	а)	X5	18	6	0.1447	0.8682
		X6	4000	4	0.1739	0.6956

Отже отримали:

$$K_K(1) = 0.5025 + 1.2565 + 1.4637 + 1.0845 + 0.8682 + 0.6956 = 5.871$$
$$K_K(2) = 0.804 + 1.0052 + 1.4637 + 1.0845 + 0.8682 + 0.6956 = 5.9212$$

Отже, другий варіант, що передбачає використання нейронних мереж дає більший результат. Тому віддаємо йому перевагу.

4.8 Економічний аналіз варіантів розробки програмного продукту

Для оцінки трудомісткості розробки спочатку проведемо розрахунок трудомісткості. Обидва варіанти мають наступні основні завдання:

Дослідження теоретичної бази прикладної області;

2.1. Розробка алгоритмічної бази для навчання мережі;

2.2. Розробка алгоритмів за допомогою фреймворків;

3. Розробка фінального програмного продукту.

Для завдання 1 (алгоритм групи складності 2, ступінь новизни А, вид використаної інформації — НДІ) T_P =36 людино-днів, $K_П$ =1.51 для НДІ, $K_{СК}$ = 1, $K_{СТ.М}$ =1.5.

$$T_1^1 = 36 * 1.51 * 1.5 = 81.54 \text{ людино-днів}$$

Для завдання 2, при реалізації варіанту 1 (алгоритм групи складності 3, ступінь новизни А, вид використаної інформації — НДІ) $T_Р = 27$, $К_П = 1.26$ для НДІ, $К_{СК} = 1$, $К_{СТ.М} = 1.4$.

$$T_1^2 = 27 * 1.26 * 1.4 = 47.628 \text{ людино-днів}$$

Для завдання 2, при реалізації варіанту 2 (алгоритм групи складності 3, ступінь новизни А, вид використаної інформації — НДІ) $T_Р = 27$, $К_П = 1.26$ для НДІ, $К_{СК} = 1$, $К_{СТ.М} = 1.3$.

$$T_2 = 27 * 1.26 * 1.3 = 44.226 \text{ людино-днів}$$

Для завдання 3 (алгоритм групи складності 2, ступінь новизни Б, вид використаної інформації НДІ) $T_Р = 27$ людино-днів, $К_П = 1.08$, $К_{СК} = 1$, $К_{СТ} = 0.8$, $К_М = 1$.

$$T_3 = 27 * 1.08 * 0.8 = 23.328 \text{ людино-днів}$$

Загальна кількість людино-днів:

$$T^1 = 81.54 + 47.628 + 23.328 = 152.496 \text{ людино-днів}$$
$$T^2 = 81.54 + 44.226 + 23.328 = 149.094 \text{ людино-днів}$$

В розробці та проведенні дослідження приймають участь 1 програміст з окладом 18 000 грн та один математик-теоретик з окладом 10 000.

Зарплата розробників становить погодинно:

$$\frac{18000+10000}{2*22*8} = 79.54 \text{ грн.}$$

Зарплата поваріантно:

$$C_1 = 79.54 * 152.496 * 8 = 97\,037 \text{ грн.}$$

$$C_2 = 79.54 * 149.094 * 8 = 94\,872 \text{ грн.}$$

Відрахування на соціальний внесок становить 22%:

$$C_1^{v} = 0.22 * 98\,330 = 21\,349 \text{ грн.}$$

$$C_2^{v} = 0.22 * 100\,495 = 20\,871 \text{ грн.}$$

Визначаємо витрати на оплату однієї машино-години, враховуючи заробітну плату програміста в розмірі 18000 грн з коефіцієнтом зайнятості 0.4:

$$C_Г = 12*М*K_З = 12 *18000* 0.4 = 86\,400 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г* (1+ K_З) = 86\,400 * (1 + 0.2) = 103\,680 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} * 0.22 = 103\,680 * 0{,}22 = 22\,809 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 22000 грн.

$$C_А = K_{ТМ*} K_{А*} Ц_{ПР} = 1.15 * 0.25 * 22000 = 6325 \text{ грн.}$$

Витрати на ремонт та профілактику можна підрахувати:

$$С_Р = К_{ТМ}*Ц_{ПР} * К_Р = 1.15 * 22000 * 0.05 = 1265 \text{ грн.}$$

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С–Д_Р)*t_З*К_В = (365 – 104 – 11 – 16) * 8 * 0.9 = 1684.8 \text{ годин}$$

Тепер рахуємо витрати на оплату електроенергії (з урахуванням ПДВ):

$$С_{ЕЛ} = Т_{ЕФ}* N_С* К_З* Ц_{ЕН} =1684.8 * 0.6* 2 * 1.75 = 3538.08 \text{ грн.}$$

Накладні витрати рахуються наступним чином:

$$С_Н = Ц_{ПР}*0.67 = 22000·0,67 =14740 \text{ грн.}$$

Річні експлуатаційні витрати:

$$С_{ЕКС} = 103680 + 22809 + 6325 + 1265 + 3538.08 + 14740 = 152357.08 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ становитиме:

$$С_{М-Г} = С_{ЕКС}/ Т_{ЕФ} = 152\ 357.08 /1684.8 = 90.43 \text{ грн/год}$$

Витрати на оплату машинного часу складають в залежності від варіанту:

$$С_М^1 =90.43 *152.496*8 =110\ 321.71 \text{ грн.}$$

$$C_M^2 = 90.43 * 149.094 * 8 = 107\ 860.56 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H^1 = 0.67 * 110\ 321.71 = 73\ 915.54 \text{ грн.}$$
$$C_H^2 = 0.67 * 107\ 860.56 = 72\ 266.57 \text{ грн.}$$

Таким чином, вартість розробки ПП та проведення дослідів складає:

$$C_1 = 97\ 037 + 21\ 349 + 110\ 321.71 + 73\ 915.54 = 302\ 623.25 \text{ грн.}$$
$$C_2 = 94\ 872 + 20\ 871 + 107\ 860.56 + 72\ 266.57 = 295\ 870.13 \text{ грн.}$$

Проведемо розрахунок техніко-економічного рівня:

$$К_{ТЕР1} = 5.871 / 302\ 623.25 = 1.94 \cdot 10^{-5},$$
$$К_{ТЕР2} = 5.9212 / 295\ 870.13 = 2.01 \cdot 10^{-5}.$$

Отже найбільш ефективним є другий варіант з коефіцієнтом техніко-економічного рівня $2.01 \cdot 10^{-5}$.

4.9 Висновки до розділу

Було проведено повний функціонально-вартісний аналіз програмного продукту, який було розроблено в рамках дипломного проекту. За допомогою технічного дослідження було виділено 2 можливих варіанти реалізації ПП. Після їх порівняння з економічної точки зору, було визначено, що 2-ий варіант реалізації є більш доцільним з техніко-економічним рівнем $2.01 \cdot 10^{-5}$. Цей варіант передбачає:

1. Використання нейронних мереж

2. Застосування готових фреймворків для розробки
3. Використання нових, простіших ігор як середовище
4. Використовувати мову програмування Python

Даний варіант виконання програмного продукту надає досить потужний функціонал, прийнятну швидкодію та гарну якість моделі.

ВИСНОВКИ

В цій роботі було розглянуто теорію ігор з комбінаторними іграми, існуючі методи для пошуку виграшної стратегії у комбінаторних іграх двох осіб.

Розглянуто приклади комбінаторних ігор, їх класифікацію, досліджена актуальність дослідження цієї області та історія розвитку цього напрямку. Було розглянуто класичні підходи теорії до задачі.

Далі було проведено огляд теорії навчання з підкріпленням, розглянуто існуючі методи навчання, проведено їх порівняння та детальний опис.

Із розгялунтих методів для реалізаціїї було обрано алгоритм DQN. Він був реалізований у вигляді програми, яка навчає агента для комбіанторної гри Прорив. У рамках дослідження було перевірено поведінку алгоритмів та якість отриманих агентів при різних початкових умовах і наданий аналіз цих результатів.

У подальшому досліджені планується:

дослідження більш складної задачі, в рамках якої агент грає у більш широкий клас ігор: ігри з неповною інформацією, ігри з нічиєю тощо;

дослідження застосування методів багатоагентного навчаннння з підкріпленням для цих задач (наприклад метод DeepCFR мінімізації);

Розробка більш узагальненої бібліотеки для поліпшення дослідження різних ігор за допомогою методів навчання з підкріпленням.

РЕКОМЕНДАЦІЇ

Дослідження пошуку стратегії в різних іграх є одним із головних досліджень у сфері штучного інтелекту. Назараз проводиться активне дослідження і розробка узагальнених алгоритмів для пошуку стратегій у різних іграх та середовищах. У подальшому досліджені можна провести:

дослідження більш складної задачі, в рамках якої агент грає у більш широкий клас ігор: ігри з неповною інформацією, ігри з нічиєю тощо;

дослідження застосування методів багатоагентного навчанння з підкріпленням для цих задач (наприклад метод DeepCFR мінімізації);

Розробку більш узагальненої бібліотеки для поліпшення дослідження різних ігор за допомогою методів навчання з підкріпленням.

Отримані в процесі роботи та розробки результати можна застосовувати для оцінки оптимальної стратегії у реальних нових іграх, що є важливим практичним застосуванням. Теоретичні напрацювання можна використати для подальшого розвитку розробки стратегій в іграх та розробки вдосконалених рішень для такої задачі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Berlekamp E. The Economist's View of Combinatorial Games. Games of No Chance MSRI Publications. 1996. Vol 29. P.33-43.
2. Larson U., Nowakowski R., Carlos S. Absolute combinatorial game theory. arXiv. 2016. P.12-21 URL: https://arxiv.org/abs/1606.01975
3. Bouton C.L. Nim, A Game with a Complete Mathematical Theory. Annals of Mathematics Second Series. 1901. Vol. 3, No. 1/4. P.35-39.
4. Campbell M.S., Hoane Jr. A., Hsu F. Deep Blue. Artificial Inteligence. 2002. Vol. 134. P.57-83.
5. Silver D., Huang A., Maddison C. Mastering the game of Go with deep neural networks and tree search. Nature. 2014. Vol. 529(7587), P. 484-489.
6. Silver D. Mastering the Game of Go without Human Knowledge. Nature. 2017. Vol. 550(7676). P. 354–59.
7. Silver D. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play. Science. 2018. Vol. 362(6419). P. 1140–1144.
8. Gelly S. The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions. Communications of the ACM. 2012. Vol. 55(3). P. 106–113.
9. Cowling P.I., Powley E.J., Whitehouse D. Information Set Monte Carlo Tree Search. IEEE Transactions on Computational Intelligence and AI in Games. 2012. Vol. 4. No. 2. P. 120-143.
10. Фролов И. С. Введение в теорию комбинаторных игр. Простейшие комбинаторные игры. Матем. обр. 2012. № 3(63). С. 38-52.
11. Sutton R., Barto A. Reinforcement learning: an introduction. Second edition / MIT Press. Cambridge. 2018. 44 c.
12. Blackwell D. Discrete dynamic programming. The Annals of Mathematical Statistics. 1962. P. 719-726.
13. Feinberg E., Kasyanov P., Zgurovsky M. Convergence of value iterations for total-cost MDPs and POMDPs with general state and action sets. IEEE

Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL). 2014. P. 1-8.

14. Teng TH. Self-Regulating Action Exploration in Reinforcement Learning. Procedia Computer Science. 2012. Vol. 13. P. 18–30.
15. Sledge I., Principe J. Balancing exploration and exploitation in reinforcement learning using a value of information criterion. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE. 2017. P. 2816–2820.
16. Mnih V., Human-Level Control through Deep Reinforcement Learning. Nature. 2015. Vol. 518(7540). P. 529–533.
17. Strehl A., Littman M. An analysis of model-based interval estimation for Markov decision processes. Journal of Computer and System Sciences. 2008. Vol. 74, No. 8. P. 1309-1331.
18. Pathak D. Curiosity-Driven Exploration by Self-Supervised Prediction. 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). IEEE. 2017. P. 488–89
19. Badia A.P., Piot B., Kapturowski S., Sprechmann P. Agent57: Outperforming the atari human benchmark. arXiv preprint. 2020. P.1-30 URL: https://arxiv.org/abs/2003.13350
20. Ecoffet A., Huizinga J., Lehman J. Go-explore: a new approach for hard-exploration problems. arXiv preprint. 2019. P.5-36 URL: https://arxiv.org/abs/1901.1099
21. Weber T., Racanière S., Reichert D. Imagination-augmented agents for deep reinforcement learning. arXiv. 2017. P.1-20 URL: https://arxiv.org/1707.06203
22. Nagabandi A., Kahn G., Fearing R. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. IEEE International Conference on Robotics and Automation (ICRA). 2018. P. 7559-7566
23. Van Hasselt H., Guez A., Silver D. Deep reinforcement learning with double q-learning. arXiv preprint. 2015. URL: https://arxiv.org/1509.06461

24. Schaul T., Quan J., Antonoglou I., Antonoglou D. Prioritized experience replay. arXiv preprint. 2015. https://arxiv.org/1511.05952
25. Wang Z., de Freitas N., Lanctot M. Dueling Network Architectures for Deep Reinforcement Learning. ArXive-prints. 2015. URL: https://arxiv.org/abs/1511.06581
26. Fortunato M., Gheshlaghi Azar M., Piot B., Menick J. Noisy Networks For Exploration. In International Conference on Learning Representations. 2018. URL: https://openreview.net/forum?id=rywHCPkAW
27. Sutton R., McAllester D., Singh S., Mansour Y. Policy gradient methods for reinforcement learning with function approximation. In Advances in Neural Information Processing Systems (NIPS). 1999. MIT Press, Cambridge, MA, USA. P. 1057–1063.
28. Baxter J., Bartlett P. Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research. 2001. Vol. 15. P. 319-350.
29. Mnih V., Badia A.P., Mirza M., Graves A. Asynchronous methods for deep reinforcement learning. arXiv preprint. 2016. URL: https://arxiv.org/abs/1602.01783
30. Wu, Y., Mansimov, E., Liao, S., Grosse, R., Ba J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. arXiv preprint. 2017. URL: https://arxiv.org/abs/1708.05144
31. Degris T., White M., Sutton R. Off-Policy Actor-Critic. arXiv preprint. 2012. URL: https://arxiv.org/abs/1205.4839
32. Lanctot M. OpenSpiel: A framework for reinforcement learning in games. arXiv preprint. 2019. URL: http://arxiv.org/abs/1908.09453

ДОДАТОК А

Програмний код

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

from absl import app
from absl import flags
from absl import logging
import numpy as np
import tensorflow.compat.v1 as tf

from open_spiel.python import rl_environment
from open_spiel.python.algorithms import dqn
from open_spiel.python.algorithms import random_agent

config = tf.ConfigProto(log_device_placement=True)

checkpoint_dir= "/content/drive/My
Drive/diploma_checkpoints/breakthrough/dqn_18x18"
num_train_episodes= 100000
eval_every = 1000

# DQN model hyper-parameters
hidden_layers_sizes = [64, 64, ]
replay_buffer_capacity = int(1e5)
batch_size= 32

def eval_against_random_bots(env, trained_agents, random_agents,
num_episodes):
        """Evaluates `trained_agents` against `random_agents` for
`num_episodes`."""
        num_players = len(trained_agents)
        sum_episode_rewards = np.zeros(num_players)
        for player_pos in range(num_players):
        cur_agents = random_agents[:]
        cur_agents[player_pos] = trained_agents[player_pos]
        for _ in range(num_episodes):
                time_step = env.reset()
                episode_rewards = 0
                while not time_step.last():
                        player_id = time_step.observations["current_player"]
                        if env.is_turn_based:
```

```
                                agent_output = cur_agents[player_id].step(
time_step, is_evaluation=True)
                                action_list = [agent_output.action]
                        else:
                                agents_output = [
                                agent.step(time_step, is_evaluation=True) for agent in
                        cur_agents
]
                                action_list = [agent_output.action for agent_output in
                        agents_output]
                time_step = env.step(action_list)
                episode_rewards += time_step.rewards[player_pos]
                sum_episode_rewards[player_pos] += episode_rewards
        return sum_episode_rewards / num_episodes

game = "breakthrough"
num_players = 2

env_configs = {"columns": 8, "rows": 8}
env = rl_environment.Environment(game, **env_configs)
info_state_size = env.observation_spec()["info_state"][0]
num_actions = env.action_spec()["num_actions"]

# random agents for evaluation
random_agents = [
random_agent.RandomAgent(player_id=idx, num_actions=num_actions)
for idx in range(num_players)
]

"""Train agent"""

losses = [[],[]]
rewards_history = [[],[]]

with tf.Session(config=config) as sess:
        hidden_layers_sizes = [int(l) for l in hidden_layers_sizes]
        agents = [
        dqn.DQN(
session=sess,
player_id=idx,
state_representation_size=info_state_size,
num_actions=num_actions,
hidden_layers_sizes=hidden_layers_sizes,
replay_buffer_capacity=replay_buffer_capacity,
batch_size=batch_size) for idx in range(num_players)
```

```
]
        saver = tf.train.Saver()
        sess.run(tf.global_variables_initializer())
        for ep in range(num_train_episodes):
                if (ep + 1) % eval_every == 0:
                        r_mean = eval_against_random_bots(env, agents,
                random_agents, 1000)
                        print("{} Mean episode rewards {}".format(ep + 1, r_mean))
                        rewards_history[0].append(r_mean[0])
                        rewards_history[1].append(r_mean[1])
                        saver.save(sess, checkpoint_dir, ep)
                time_step = env.reset()
                while not time_step.last():
                        player_id = time_step.observations["current_player"]
                        if env.is_turn_based:
                                agent_output = agents[player_id].step(time_step)
                                action_list = [agent_output.action]
                        else:
                                agents_output = [agent.step(time_step) for agent in
                                agents]
                                action_list = [agent_output.action for agent_output in
                        agents_output]
                        time_step = env.step(action_list)

                # Episode is over, step all agents with final info state.
for agent in agents:
                agent.step(time_step)
                losses[0].append(agents[0].loss)
                losses[1].append(agents[1].loss)



import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


fig,ax = plt.subplots(figsize = (12,8))
plt.title('Estiamte rewards')
plt.plot(rewards_history[0], label = '1st player')
plt.plot(rewards_history[1], label = '2nd player')

formatter = ticker.FormatStrFormatter('%d*1e3')
ax.xaxis.set_major_formatter(formatter)

for tick in ax.xaxis.get_major_ticks():
```

```
tick.label1.set_visible(True)
tick.label2.set_color('green')

plt.legend()
plt.show()
```

ДОДАТОК Б

Ілюстративний матеріал для доповіді

Методи навчання з підкріпленням в комбінаторних іграх двох учасників

Виконав:
Студент 4-го курсу
групи КА-61
Єлісєєв Владислав

Керівник:
Д-р фіз.мат. наук, доцент
Ігнатенко О. П.

Вступ

Мета роботи: дослідити останні досягнення науки у сфері навчання з підкріпленням, розглянути можливості застосування методів навчання з підкріпленням до задач пошуку стратегій гри у комбінаторних іграх.

Об'єкт дослідження: комбінаторні ігри двох осіб з повною інформацією, де складність пошуку стратегії полягає у великий кількості стратегій та траєкторій гри, що ускладнює пошук оптимальної стратегії.

Вступ

Предмет дослідження: методи глибокого навчання з підкріпленням для вирішення задачі пошуку виграшної стратегії у комбінаторних іграх та їх застосування на конкретних прикладах.

Методи дослідження: розробка програмного продукту для навчання агентів, що приймає рішення, у комбінаторних іграх, розробка виконана за допомогою мови програмування Python і фреймворків TensorFlow та OpenSpiel.

Актуальність

Актуальність дослідження: Комбінаторні ігри часто мають відносно прості правила, завдяки чому вони часто набувають популярності.

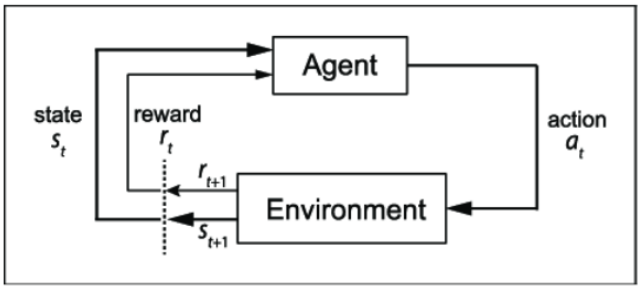
Також комбінаторні ігри є дуже привабливими з точки зору розвитку штучного інтелекту. Через їх складність, величезну кількість можливих станів та можливих дій, на сучасному етапі розвитку інформаційних технологій неможливо побудувати напряму систему прийняття рішень, яка б брала до уваги всі комбінації та можливі дії гравців, а отже виникає задача розробки систем штучного інтелекту для них.

Постановка задачі

В роботі поставлено наступну задачу:

- дослідити наявні методи пошуку виграшних стратегій у комбінаторних іграх;
- розглянути можливість застосування методів глубокого навчання з підкріпленням до задачі пошуку виграшної стратегії у комбінаторних іграх;
- визначити методи, придатні для застосування у даній задачі, і застосувати їх на прикладі конкретної комбінаторної гри.

Задача навчання з підкріпленням



Модель взаємодії середовища та агенту у навчанні з підкріпленням

Навчання з підкріпленням

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$$

$$Q(s_t, a_t) = \sum_{t'=t}^{T}\mathbb{E}_{\pi_\theta}[R(s_{t'}, a_{t'})|s_t, a_t]$$

$$V(s_t) = \mathbb{E}_{a_t\sim\pi_\theta(a_t,s_t)}[Q(s_t, a_t)]$$

Метод DQN

Deep Q-Networks (DQN)

- Value-based RL algorithm
- Learn a Q-Value function obeys a Bellman Equation

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma Q^*(s', a')|s', a']$$

- Loss Function

$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a', \theta') - Q(s, a, \theta))^2]$$

Ігрові середовища для навчання

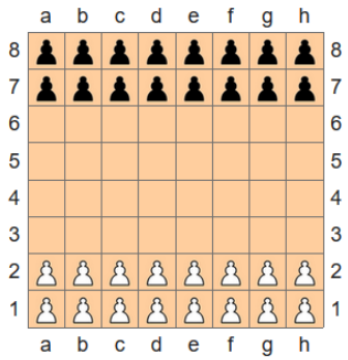
DeepMind

2020-1-1

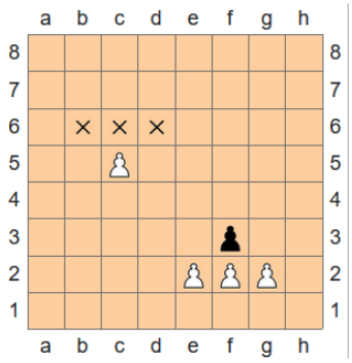
OpenSpiel: A Framework for Reinforcement Learning in Games

У 2019 році компанія DeepMind випустила OpenSpiel - це сукупність середовищ та алгоритмів для дослідження навчання з підкріпленням та пошуку/планування у іграх.

Середовище - гра “Прорив”



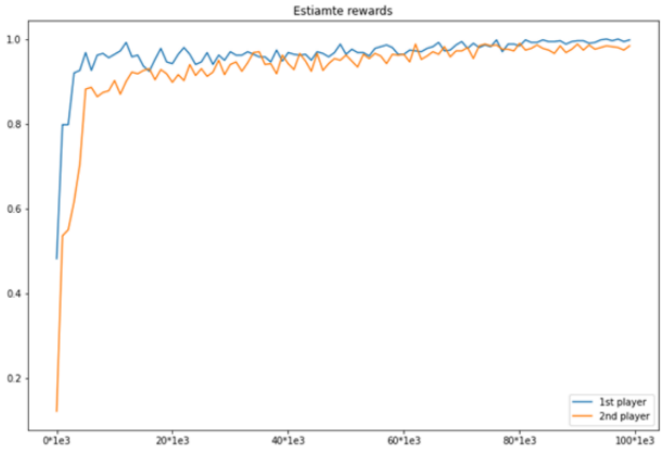
Початковий стан гри



Можливі дії

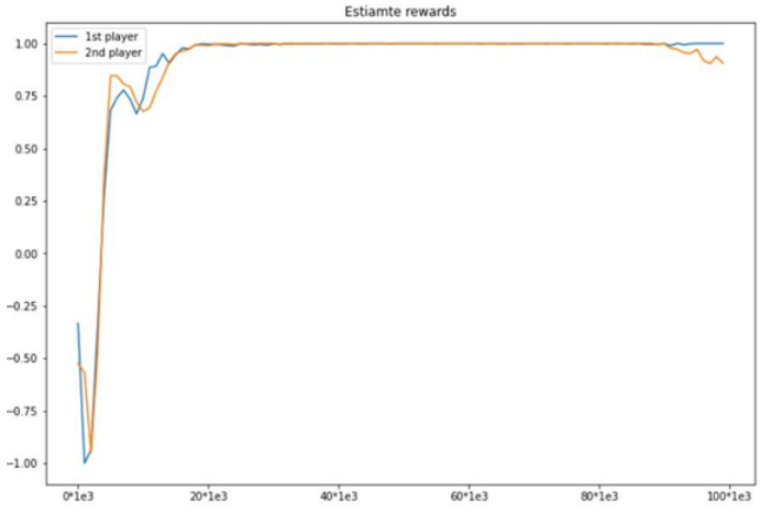
Результати роботи

4x4 дошка, 4 фігури для кожного гравця.



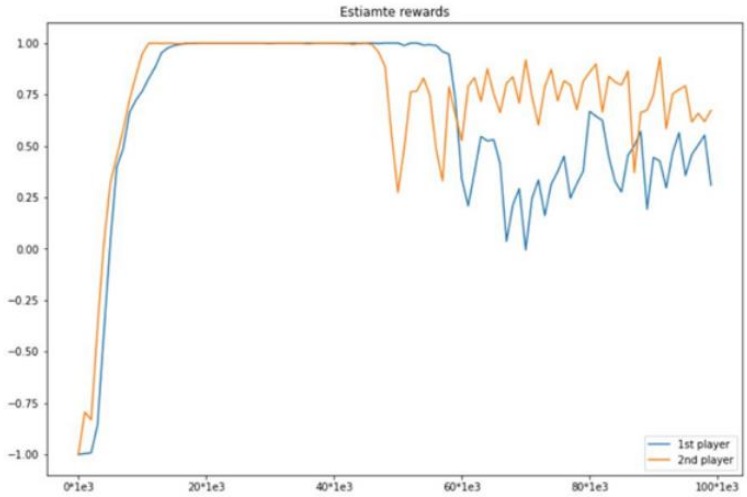
Результати роботи

5x10 дошка, 10 фігур для кожного гравця.



Результати роботи

8x8 дошка, 16 фігур для кожного гравця.



Висновки

В роботі було досліджено наявні методи пошуку стратегій а також можливість застосування методів глибокого навчання з підкріпленням у задачі пошуку виграшної стратегії. Під час виконання роботи було розроблено програму, яка навчає за допомогою методів навчання з підкріпленням (DQN) навчає двох агентів для гри “Прорив”, які показують високі результати у грі проти випадкових гравців.

Перспективи досліджень

У подальшому досліджені планується:

- дослідження більш складної задачі, в рамках якої агент грає у більш широкий клас ігор: ігри з неповною інформацією, ігри з нічиєю тощо;
- дослідження застосування методів багатоагентного навчанння з підкріпленням для цих задач (наприклад метод DeepCFR мінімізації);
- Розробка більш узагальненої бібліотеки для поліпшення дослідження різних ігор за допомогою методів навчання з підкріпленням.

Дякую за увагу!